CS56—Midterm Exam 2—Question 1

E02, Q01, W16, Phill Conrad, UC Santa Barbara 02/10/2016

HAND THIS IN WITH YOUR EXAM.
YOU MAY USE THIS FOR SCRATCH WORK, BUT ALL ANSWERS SHOULD BE ON YOUR EXAM PAPER.

Name: _____

Umail Address: _____@ umail.ucsb.edu

Code for class `Foo`, `Bar`, and `Fum`

```
/*  1 */ public class Foo {
/*  2 */
/*  3 */    public int mysteryNumber;
/*  4 */
/*  5 */    void doThing() {
/*  6 */             System.out.println("x");
/*  7 */    }
/*  8 */
/*  9 */    public static void main(String [] args) {
/* 10 */      Foo foo = new Foo();
/* 11 */      Bar bar = new Bar();
/* 12 */      Fum fum = new Fum();
/* 13 */      System.out.println("Hello");
/* 14 */      /* this is line 14 */
/* 15 */    }
/* 16 */  }
/* 17 */
/* 18 */ class Bar extends Foo {
/* 19 */
/* 20 */    public boolean mysteryBoolean;
/* 21 */
/* 22 */    void doThing() {
/* 23 */        System.out.println("y");
/* 24 */    }
/* 25 */ }
/* 26 */
/* 27 */ class Fum extends Foo {
/* 28 */    public String mysteryString;
/* 29 */
/* 30 */    void doOtherThing() {
/* 31 */        System.out.println("z");
/* 32 */    }
/* 32 */ }
```

Javadoc for java.util.ArrayList is on the other side

java.util.ArrayList<E>

boolean	**add**(E e) Appends the specified element to the end of this list.
void	**add**(int index, E element) Inserts the specified element at the specified position in this list.
boolean	**addAll**(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
boolean	**addAll**(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this ArrayList instance.
boolean	**contains**(Object o) Returns true if this list contains the specified element.
void	**ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
E	**get**(int index) Returns the element at the specified position in this list.
int	**indexOf**(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	**isEmpty**() Returns true if this list contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this list in proper sequence.
int	**lastIndexOf**(Object o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<E>	**listIterator**() Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<E>	**listIterator**(int index) Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
E	**remove**(int index) Removes the element at the specified position in this list.
boolean	**remove**(Object o) Removes the first occurrence of the specified element from this list, if it is present.
boolean	**removeAll**(Collection<?> c) Removes from this list all of its elements that are contained in the specified collection.
protected void	**removeRange**(int fromIndex, int toIndex) Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.
boolean	**retainAll**(Collection<?> c) Retains only the elements in this list that are contained in the specified collection.
E	**set**(int index, E element) Replaces the element at the specified position in this list with the specified element.
int	**size**() Returns the number of elements in this list.
List<E>	**subList**(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.

CMPSC 56 E02

[redacted]

TOTAL POINTS

70 / 70

QUESTION 1

Java Swing 30 pts

1.1 (a) Purpose of FooPanel? (5 / 5)

+ 5 Correct: a widget that implements a GUI, or contains an interactive widget (e.g. a button) that needs a callback routine when it is clicked (or activated in some way).

+ 5 Alternative Correct Answer: to receive events and perform the desired actions in response.

+ 3 Partial Credit: Answer that is correct, but focuses on what an "interface" is, missing the point that ActionListener has a specific role in Java Swing. The question was getting at the "purpose" of class FooPanel in a specific sense, not an abstract sense.

+ 3 Partial Credit: Answer focuses on what an "inner class" is, missing the point that ActionListener has a specific role in Java Swing. The question was getting at the "purpose" of class FooPanel in a specific sense, not an abstract sense.

1.2 (b) What Java keyword? (5 / 5)

+ 5 Correct: this

+ 0 Incorrect: ActionEvent or ActionEvent object

+ 0 Incorrect: inner class

+ 0 Incorrect: Listener object

+ 0 Incorrect: public class FooPanel implements ActionListener

+ 0 Incorrect: Listener or Listeners

+ 0 Incorrect: Instance Variable

+ 2 Partial Credit: *this (The prefix * is a C++ thing.)

+ 0 incorrect: dot operator

+ 0 Incorrect: ActionListener

+ 0 Left Blank

+ 0 Incorrect: protected

+ 0 Incorrect: implements

+ 0 Incorrect: ActionHandler

+ 0 Incorrect: Event handler

+ 0 Incorrect: static

+ 0 Incorrect: widget

+ 0 Incorrect: new

+ 0 Not a rubric item, just a comment: see http://stackoverflow.com/questions/3124126/java-addactionlistenerthis for more information on this topic.

1.3 (c) Disadvantage of making FooPanel be object that implements ActionListener? (5 / 5)

+ 5 Correct: there can be only one actionPerformed method in the class. (That's a problem if you have more than two things you want to do---because you'll have to have complex code to handle all the different cases inside your one single actionPerformed method.)

+ 0 Left Blank

+ 4 PARTIAL CREDIT correct answer: It may not enforce separation of concerns or good object-oriented practices. [Awarded 4/5 because this is a good generic answer for why to make something a separate object. In this case, there is a more specific reason as well that is context dependent that we discussed in lecture.]

+ 0 Incorrect: Unable to reuse the code written in the inner class. This is incorrect, because part c is not discussing the use of an inner class at all.

+ 0 Incorrect answer: "It only applies to events contained within the FooPanel class. If we have another class with the same event, it will not work." OR "You are not allowed to use the ActionListener outside of the FooPanel class". Neither of those is true. We "could" use an instance of this class to be an ActionListener for some other class' event. So it is

possible. But even if it were true that we couldn't (and that's NOT true), it is very unlikely that we would want to. An ActionListener in a GUI needs to make changes to the state of elements of a particular GUI. When you press a button, or scroll a scroller, or click a mouse, something is supposed to "happen". And making that thing happen likely requires access to the internal state of the FooPanel. And that fact pretty much negates the whole point you are trying to make, i.e. that "reuse" is something desirable here.

+ 0 Incorrect Ansswer: "If somewhere else in the program you need the same ActionListener, your code will not be DRY." That "seems" like a plausible answer---in general, we try to avoid duplicate code to make things DRY. But in this case, that's not the problem. . An ActionListener in a GUI typically needs to make changes to the state of elements of that GUI. When you press a button, or scroll a scroller, or click a mouse, something is supposed to "happen". And making that thing happen likely requires access to the internal state of the FooPanel. It is not likely that you are going to be able to reuse that code anywhere else in your program.

+ 0 Incorrect Answer: "This will make the FooPanel both extends JPanel and implements ActionListener, which may cause writing method of same name." So, I'm not sure what problem you are trying to describe here--- "may cause writing method of same name" isn't very precise. Same name as what? In any case having a class that both. extends JPanel and implements ActionListener is a perfectly legal thing to do, and causes no naming conflicts.

+ 2 Partial credit answer: "You have to create an instance of the FooPanel class itself within the FooPanel class. It's more complicated and its' not good for data encapsulation. There can only be one implementation." That last sentence, i.e. "there can only be one implementation" is what saved you from a zero credit answer---because that is true, though you didn't really explain why that's a problem. The first part is just false---you don't need a separate instance. The keyword "this" is a reference back to the instance itself. So, there is only one instance, and the data encapsulation is just fine---perfect, in a sense, since we are dealing only with a single object.

+ 0 Incorrect Answer. "This limits what the FooPanel class is able to do because each class is made so that it specializes at one job." This is incorrect. Implementing ActionListener doesn't "limit" what FooPanel can do. On the contrary, the essence of implementing an interface is that it provides a marker of some additional capability that the class CAN do. It expresses the idea "this class can do this thing, because it has all the necessary methods to do it". The second part of your sentence: "because each class is made so that it specializes at one job", describes an important design goal of a good object oriented system. But it is not the correct answer to THIS question.

+ 0 Incorrect answer: "There might not be a visual indicator of what or where the ActionListener is for the user". Incorrect because a "user" would never need to have any visual indicator of an ActionListener--its entirely an internal code construct, not a user-facing thing. If you mean "programmer" rather than "user", it's still incorrect. It will be visible from the fact that FooPanel class will say "implements ActionListener" right at the top, and the fact that the addActionListener() method for any widget with an ActionListener will take the parameter "this". Both of those are visual indicators of what or where the ActionListener is for the programmer.

+ 0 Not a rubric item, just a comment: see http://stackoverflow.com/questions/3124126/java-addactionlistenerthis for more information on this topic.

+ 0 Incorrect: "The FooPanel class can ONLY do what the ActionListener interface can do, no more, no less". OR "FooPanel would not be able to make method calls with a normal object functionality. FooPanel will only be used for waiting until the user makes an action. This means you cannot [illegible... have? hide?] an ActionEvent in this class definition." That is not correct. FooPanel can be a JPanel that has any

kind of functionality that a JPanel would normally have, PLUS it can also be an ActionListener. All we have to do is add an actionPerformed method to the class.

+ 3 PARTIAL CREDIT: "If FooPanel implements ActionListener, there can only ever be 1 interactive button/widget in the Panel..." That's not correct. There can be several. The problem isn't that we can't have more than one--its that we have to stuff all of that code for handling, say, a Button, a TextArea and a slider into a single method that has to complicated logic (e.g. a series of if tests) to determine which kind of event happened. Still, 3/5 because you are at least on the right track with this line of thinking.

+ 0 Incorrect: "That will let FooPanel implement a lot [sic] useless methods and in this case FooPanel needs to implement all the methods in ActionListener which are not what we want for FooPanel.". So, ActionListener has only one method (actionPerformed) and it really isn't a big deal to implement it. If we only need one action for a single widget, it may be less work to add this one method than to create a separate class.

1.4 (d) Disadvantage of separate class? (5 / 5)

+ 5 Correct: the separate class does not have access to the private members of FooPanel. (Which it will likely need in order to make things happen in the GUI).

+ 2 Partial credit: bulky code involved with separate classes for separate ActionListeners...vague or fails to mention idea that event handlers need access to instance variables.

+ 0 Incorrect: fails to identify a disadvantage of separate classes

+ 0 Left Blank

+ 3 Partial Credit: Correctly indicates that it is more work, with this architecture, to make changes to FooPanel from inside the ActionListener, but doesn't clearly explain why.

1.5 (e) Briefly describe third approach to ActionListener (not self, not separate class) (5 / 5)

+ 5 Correct: Making an inner class that implements ActionListener is the third approach.

+ 5 ALTERNATIVE correct answer--if "anonymous inner class" used as answer to part (e). This lists various advantages of an anonymous inner class.

+ 3 Partial credit: correctly identifies inner class, but doesn't mention that the inner class will be the one implementing ActionListener

+ 2 Partial credit: answer--"have a method in FooPanel implement ActionListener..." incorrectly says method instead of inner class, but interfaces must be implemented by a class

+ 0 Incorrect: doesn't identify a third approach on how to implement ActionListener

+ 0 blank

1.6 (f) Advantages of third approach (5 / 5)

+ 5 Correct: One or more inner class objects can be used to implement one or more ActionListeners, and each of those will have full access to the outer class' instance variables.

+ 5 ALTERNATIVE correct answer--if "anonymous inner class" used as answer to part (e). This lists various advantages of an anonymous inner class.

+ 3 Partial credit: correctly includes access to instance variables, but also should discuss how inner classes allow you to have multiple ActionListeners to handle multiple events

+ 3 Partial credit: correctly states that inner classes allow you to have multiple ActionListeners to handle multiple events, but doesn't say that inner classes have access to instance variables

+ 1 Partial credit: vague or incomplete response that fails to identify advantages of inner classes

+ 0 Incorrect: identifies disadvantages of inner classes

+ 0 Incorrect: doesn't identify an advantage of inner classes

+ 0 blank

QUESTION 2

Foo, Bar, Fum (inheritance) 20 pts

2.1 What methods can be invoked on foo? (5 / 5)

+ 5 **Correct: doThing() lines 5-7 (ok to say line 5 too).**

+ 0 OK to include main, since foo.main(args) will, technically, compile (I tried it). Also ok to omit this from answer, since we typically would not do this.

- 1 Directions indicated to include the line number in your answer.

2.2 What public data members can follow bar. (5 / 5)

+ 5 **Correct: mysteryBoolean and mysteryNumber**

- 2 -2 for including doThing() in the answer along with mysteryBoolean and mysteryNumber, since it is not a data member. Also, doThing() returns void, so an invocation of it cannot be used as argument of System.out.println()

+ 0 no points for answer doThing() if it is the only thing included in the answer. doThing() is not a data member.

+ 2 2 points for only including mysteryNumber, and not including mysteryBoolean.

+ 2 2 points for only including mysteryBoolean, and not including mysteryNumber.

2.3 Methods can be invoked on fum? (5 / 5)

+ 5 **[NOTE TYPO CORRECTION given to all students: "object foo" should read "object fum".] Correct: doThing, line 5 (or 5-7) and doOtherThing, line 30, (or 30-32)**

+ 0 Ok to include main; technically, it can be invoked on object fum; but not necessary for full credit.

+ 2 Partial Credit for mentioning only doOtherThing from lines 30-32 and leaving out doThing from lines 5-7, which is inherited.

2.4 Data members for fum? (5 / 5)

+ 5 **Correct: mysteryNumber and mysteryString**

- 2 -2 for including doThing and/or doOtherThing in the answer along with mysteryBoolean and mysteryString, since those are methods (member functions), not data members. Also, they return void, so an invocation of them cannot be used as argument of System.out.println()

+ 2 Partial credit for answer "mysteryString", omitting mysteryNumber which should also be included, as it is inherited.

+ 0 No credit for answer consisting only of doThing and doOtherThing, since those are methods (member functions), not data members. Also, they return void, so an invocation of them cannot be used as argument of System.out.println()

QUESTION 3

3 Two main categories of exceptions in Java. Names of two kinds, and reason why. (10 / 10)

+ 4 **4 of the 10 points are earned for correctly IDENTIFYING THE TWO CATEGORIES.. One category is Unchecked exceptions (which inherit from java.lang.RuntimeException) and the other is Checked exceptions, which inherit from java.lang.Exception. It is acceptable if the official names of these categories aren't given but it is clear that they are being referred to via the base class such exceptions have in common (RuntimeException / RuntimeExceptions without a space, Exception capitalized). "Thrown Exceptions" is not acceptable as all exceptions are thrown.**

+ 3 **3 of the 10 points are earned for correctly/completely identifying PURPOSE OF RUNTIME/UNCHECKED EXCEPTIONS. These indicate a problem with code logic. try/catch blocks for these are not required because they would clutter the code logic.**

+ 3 **3 of the 10 points are earned for correctly/completely identifying PURPOSE OF CHECKED EXCEPTIONS. These indicate a problem that the programmer cannot prevent, should anticipate, and should either handle with a try/catch block, or report to the caller by declaring that the method may throw the exception.**

+ 0 No points for incorrect explanation of Checked

exceptions.

+ 0 No points for incorrect explanation of Unchecked exceptions.

+ 0 Your explanation of the difference is that it is a distinction between catching the problem at compile time vs. run time, but that is not correct. Both of these happen at run time. The name is a bit misleading, I acknowledge---that's all the more reason to be sure you learn the correct distinction.

+ 2 Partial credit for an identification of the two categories that is incorrect, or only partially correct. Here are some examples of incorrect pairs of names: (1) IOExceptions vs. Exceptions (2) Runtime Exceptions vs. User-Defined Exceptions (3) runtime exceptions vs. undefined exceptions (4) RunTime Exceptions vs. Compiler Exceptions, (5) the first type is RuntimeExceptions, and the second type is the kind that uses the try/catch block (6) RuntimeException vs. CriticalException (7) RuntimeExceptoin and self-declared exceptions. Examples of correctly distinguishing are (1) checked vs. unchecked exceptions (2) Runtime exceptions vs. Checked Exceptions. (3) "RuntimeException" vs. "Regular plain-old Exception". (This last one isn't the "best" way to describe it, but I would accept as long as the explanations are correct.)

+ 2 Partial Credit for identifying the dichotomy between Runtime Exceptions and the other kind, but not indicating how the latter kind have to be handled in code (caught, or declared to be thrown.)

+ 0 Click here to replace this description.

QUESTION 4

4 Collection<? extends E> ... what does this mean? (10 / 10)

+ 10 Correct: c is an object of a class that implements the Collection interface, and it is a collection of either objects of type E, or of objects from classes or interfaces that extend (inherit from) type E. That explanation is sufficient for full credit... but to explain further with an example: if we have an instance of ArrayList<Student> and Undergrad extends Student, then c could be any of the following types: ArrayList<Student>, ArrayList<Undergrad>, PriorityQueue<Student>, PriorityQueue<Undergrad>, etc. Subject to deductions below. Note that E doesn't have to be a class. It could also be an interface.

+ 5 Partial Credit: Answer that is correct but incomplete--that says nothing wrong, but doesn't get to the point about what the <? extends E> part means.

+ 5 Partial Credit: "c is a collection of elements of any type... The reason that c can contain any type is because it is a collection of type E, which is a template parameter, meaning E can be of any type.". Yes, but no. E in this case is already bound. It is a specific type, the type that THIS ArrayList<E> instance is a collection of. <? extends E> actually RESTRICTS the type of what c can be a collection of. It restricts it to being only elements of type E, or any class/interface that extends E.

+ 5 Partial Credit "c is an instance of a class that implements an interface that extends the interface Collection<E>. " OR "any object of a class that extends classes that implement the Collection interface". NO.. If that were our intent, we might write <T extends Collection<E>> boolean addAll(T c). Or we could simply write: boolean addAll(Collection<E> c). Your answer seems to indicate that the ? extends is about extending Collection, but that is NOT the case. The extending is NOT of Collection, but of E itself.

- 2 -2 deduction from full credit answer for imprecise language: "elements that are of class E or extends class E as its parameter". What you mean to say is "elements that are of class/interface E, or any class/interface that extends E as its parameter". Those extra words are important to be precise in your meaning.

+ 5 Imprecise answer: "Collection<? extends E> c means that it could be a subclass of the given data type". This is too imprecise for full credit. What does "it" refer to? Does it refer to c or E? I can't tell.

And the given data type? What does that refer to? Collection<? extends E>? E? Without knowing this, I can't tell if your answer is correct or not. Therefore, it cannot earn full credit.

+ 5 Partial credit: Any answer that indicates that Collection<? extends E> c means that "c is any object that implements Collection, where it can be a collection of any type of object at all", i.e. there is no restriction on what type of object can be in the Collection. NO. E is already "bound" to whatever this PARTICULAR ArrayList<E> is a collection of. So, not ANY object. c is a Collection of objects of class or interface E, or any class or interface that extends E, where E is something *specific*, the SPECIFIC class or interface that this ArrayList is an ArrayList of. For example, if E is Animal, and Dog extends Animal then <? extends E> means that c can be a Collection<Animal> or a Collection<Dog>

+ 5 Partial Credit: E is of whatever type the ArrayList is. c is an interface that extends whatever type the ArrayList is. If it is an ArrayList<Integer> then c will be a Collection that extends Integer. NO.

+ 5 Partial credit: c is a type of Collection that extends E. NO. c is a type of Collection that contains inside it instances of E, or any class/interface that extends E. The Colection does NOT extend E. Not in any way.

+ 5 Partial credit: T"his means that the parameter c must be compatible with the elements of the ArrayList. By this, I mean that 'c' must be an instance of a class tht is the supertype E, or a subtype of E." Almost there, but NO. What you should have said is "This means that the parameter c __is a Collection of elements that___ must be compatible with ..." You left out those crucial words "is a Collection of elements that..."

+ 5 Partial credit: "c can be any class that extends E". c could also be an interface, and it must implement Collection interface

- 3 Collection is an interface so it must be implemented, it is not a super class that can be extended

+ 5 "The means that any object that extends Collection can be put in as an argument and that Collections ? will wrap around the object E so for example if ArrayList<Dog> is put as an argument then ? will wrap around Dog and allow the function to manipulate Dog objects." Incorrect. The "extends" keyword here is NOT about extending Collection. If ArrayList<Dog> were the context, than E is Dog, and what we have for c is a Collection of Dog or a collection of some object that extends Dog.

+ 9 9/10, for mostly correct answer. "This is a polymorphic argument. Essentially a polymorphic argument is when a super class is a parameter and one can pass a subclass through this parameter." [So far so good.] "In our case, addAll can take anything that is a subclass of collections, such as Stack, ArrayList, etc. [Ooops... not subclass. We noted that Collection is an interface, so we should have said anything that *implements* Collection. Continuing...] "The <? extends E> ensures that the argument is the same object or a subclass of the object that the ArrayList holds. YES.

+ 0 No credit: "This means that c can be any class that extends Object. In Java, primitives are not objects." That is incorrect. The type expression constrains the type of what c can be far more narrowly than "any class that extends Object", which is the same as saying "anything except a primitive". The statement "In Java, primitives are not objects" is correct, but that's not relevant to the problem, so no partial credit for that.

+ 5 Partial Credit: "It means that an ArrayList can add any object E or any collection of object E, or a subclass of E. The E object is any object or collection of objects that user defines it to be." Not exactly. We can't pass an object of type E or a subclass of E to this addAll method. We can only pass a Collection of objects of type E, or a Collection of a objects of some type that is a subclass of E.

+ 5 Incorrect/imprecise answer: "That this method works for all the classes that implement the collection interface. The type of collection must extend E..." [Ooops. Not the type of collection (e.g. ArrayList,

HashSet, PriorityQueue, Stack) must extend E, but the thing that c is a collection OF must be E or extend E. You are getting 5/10 for the benefit of the doubt that you meant to refer to the "thing that c is a Collection of", but your answer isn't precise enough to get full credit. Continuing...] " ... must be an element type (object references) but cannot be of primitive type. e.g. c cannot be of type ArrayList<int> but can be of type ArrayList<Integer>." So that last part is true, but "vacuously" true, since it is true of any parameterized type in Java. That part has nothing to do with the question asked, so you get no partial credit for it.

+ 5 Incorrect answer: "First, c is a specified collection, and inside the brackets, it stands for iterating all elements in this collection because addAll will append all of the elements." So, while it is true that addAll will append all of the elements, and c is a Collection, it is NOT true that the part in brackets (i.e. <? extends E> means to iterate over the collection.

1

Page: 1 Nam[redacted]

CS56—Midterm Exam

E02, W16, Phill Conrad, UC Santa Barbara

Monday, 02/29/2016

Name: [redacted]

Umail [redacted]

- Please write your name **above AND AT THE TOP OF EVERY PAGE**
- Please put your pages **in order, facing the same way.**
  - All the odd pages have dots (•); these should be upper right, and facing up.
  - All the even numbered pages have crosses (×) at upper right and should be facing down.
- Be sure you turn in every page of this exam.
  - Each of the pages is numbered (e.g. Page 1, Page 2, etc.)
  - The last page clearly says "End of Exam".
- This exam is **closed book, closed notes, closed mouth, cell phone off**
- You are permitted **one sheet of paper** (max size 8.5x11") on which to write notes
- This sheet will be collected with the exam, and might not be returned
- Please write your name on your notes sheet

On your handout, on the back first method

The blacked out part should read:

boolean	add(E e)
	Appends the specified element to the end of this list.

1. In Java Swing applications, we sometimes need an object that implements the `ActionListener` interface. Suppose that this situation arises in the context of a class called `FooPanel`.
   a. (5 pts) Given that we need an object that implements ActionListener inside of FooPanel, what is likely the purpose of class FooPanel, and why do we need an object that implements ActionListener inside of it?

   the user will be interacting with content inside the panel, and an ActionListener object can receive user events and perform the desired response

   b. (5 pts) There are three relationships that the object that implements ActionListener can have with the class FooBar. One of those, is that the object that implements ActionListener can be an instance of FooBar itself. In this case, what Java keyword is used to refer to the object that implements ActionListener?

   this

   c. (5 pts) What is the main disadvantage of making the FooPanel class itself be the object that implements ActionListener?

   Only one implementation of ActionListener can be used

   d. (5 pts) A second technique is to make a completely separate class, separate from FooPanel, that implements ActionListener. What is the main disadvantage of this approach?

   if the desired actionEvent relies on data from class FooPanel, it would be cumbersome to perform the desired response, since the new class can't access FooPanel's private members.

   e. (5 pts) There is a third approach where the object that implements ActionListener has a different relationship with FooPanel from the two already described. What is this third approach? Briefly describe it.

   (Note: in Java 8, a fourth approach is to use Lambda Functions, but those are NOT covered on this exam, and it isn't what I'm looking for here. Those will be on the final exam.)

   Having a inner class implement ActionListener

   f. (5 pts) What are the advantages of this third approach to making an ActionListener over the other two already described?

   this allows for multiple implementations of the ActionListener interface in the outer class, the inner class could access private members / methods of the outer class

Page: 3 Name: [redacted]

2. Consider the code for classes Foo, Bar and Fum on the handout. Answer the questions below about this code.

a. (5 pts) Inside the `main` routine, locate the comment that says `This is line 14`. Suppose we were to invoke a method on the object referred to by reference `foo`. Disregarding methods that are ~~inherited~~ from class `java.lang.Object`, and considering only methods defined in the code here, list the methods we could invoke on object `foo`, and for each one, indicate the line number on which it is defined.

inherited

doThing(), line 5

b. (5 pts) On line 13, there is a `System.out.println()` statement, with argument `"Hello"`. Suppose we were to replace the argument with a reference `bar`, the dot operator, and then any of the public data members that may be accessed through the reference bar. Disregarding any that might be inherited from class `java.lang.Object`, what is the complete list of data members that could follow `bar.` on this line? List them all.

mysteryNumber, line 3

mysteryBoolean, line 20

c. (5 pts) Inside the `main` routine, locate the comment that says `This is line 14`. Suppose we were to invoke a method on the object referred to by reference `fum`. Disregarding methods that are ~~inherited~~ from class `java.lang.Object`, and considering only methods defined in the code here, list the methods we could invoke on object ~~foo~~ fum, and for each one, indicate the line number on which it is defined.

inherited

doThing(), line 5

doOtherThing, line 30

d. (5 pts) On line 13, there is a `System.out.println()` statement, with argument `"Hello"`. Suppose we were to replace the argument with a reference `fum`, the dot operator, and then any of the public data members that may be accessed through the reference fum. Disregarding any that might be inherited from class `java.lang.Object`, what is the complete list of data members that could follow `fum.` on this line? List them all.

mysteryNumber, line 3

mysteryString, line 28

3. (10 pts) Briefly describe the two main categories of exceptions in Java.

Be sure that your answer includes not only the names of the two kinds of exceptions, but also the reason that there are two different categories, and how they have to be treated differently.

Describe as if you were asked during a job interview. You should include enough detail so that the interviewer knows that you are very familiar with exceptions in Java, but not so much that you are wasting the interviewer's time.

1. Runtime exceptions (unchecked exceptions)
- these stem from flaws in coding logic, and should not happen in the first place (i.e. accessing array element out of bounds)

2. Checked Exceptions
- Some situations may be "exceptional," the programmer cannot guarentee some code will work, so exceptions are declared and thrown to recover from situations that might break the program, such as trying to read from a file, which may or may not be accessible

4. (10 pts) On the reverse side of the handout, you will find the javadoc for the class `ArrayList<E>`. The third and fourth rows in this table contain the description of this method:

```
boolean addAll(Collection<? extends E> c)
```

I will tell you two additional pieces of information:

- that `Collection<E>` is an interface
- that a number of classes implement this interface, including `ArrayList<E>`, `HashSet<E>`, `PriorityQueue<E>`, and `Stack<E>`.

With that information, answer the following question.

The type of parameter `c` is given as `Collection<? extends E>`

What does this mean? Explain briefly.

? stands for wildcard, and it extends E, a placeholder for an arbitrary class. Put together, this is saying Collection holds any class that is a subclass of a given parent class

CMPSC 56 E02

[redacted]

TOTAL POINTS

60 / 70

QUESTION 1

Java Swing 30 pts

1.1 (a) Purpose of FooPanel? (5 / 5)

+ 5 **Correct: a widget that implements a GUI, or contains an interactive widget (e.g. a button) that needs a callback routine when it is clicked (or activated in some way).**

+ 5 Alternative Correct Answer: to receive events and perform the desired actions in response.

+ 3 Partial Credit: Answer that is correct, but focuses on what an "interface" is, missing the point that ActionListener has a specific role in Java Swing. The question was getting at the "purpose" of class FooPanel in a specific sense, not an abstract sense.

+ 3 Partial Credit: Answer focuses on what an "inner class" is, missing the point that ActionListener has a specific role in Java Swing. The question was getting at the "purpose" of class FooPanel in a specific sense, not an abstract sense.

1.2 (b) What Java keyword? (0 / 5)

+ 5 Correct: this

+ 0 Incorrect: ActionEvent or ActionEvent object

+ 0 Incorrect: inner class

+ 0 Incorrect: Listener object

+ 0 Incorrect: public class FooPanel implements ActionListener

+ 0 **Incorrect: Listener or Listeners**

+ 0 Incorrect: Instance Variable

+ 2 Partial Credit: *this (The prefix * is a C++ thing.)

+ 0 incorrect: dot operator

+ 0 Incorrect: ActionListener

+ 0 Left Blank

+ 0 Incorrect: protected

+ 0 Incorrect: implements

+ 0 Incorrect: ActionHandler

+ 0 Incorrect: Event handler

+ 0 Incorrect: static

+ 0 Incorrect: widget

+ 0 Incorrect: new

+ 0 Not a rubric item, just a comment: see http://stackoverflow.com/questions/3124126/java-addactionlistenerthis for more information on this topic.

1.3 (c) Disadvantage of making FooPanel be object that implements ActionListener? (5 / 5)

+ 5 **Correct: there can be only one actionPerformed method in the class. (That's a problem if you have more than two things you want to do---because you'll have to have complex code to handle all the different cases inside your one single actionPerformed method.)**

+ 0 Left Blank

+ 4 PARTIAL CREDIT correct answer: It may not enforce separation of concerns or good object-oriented practices. [Awarded 4/5 because this is a good generic answer for why to make something a separate object. In this case, there is a more specific reason as well that is context dependent that we discussed in lecture.]

+ 0 Incorrect: Unable to reuse the code written in the inner class. This is incorrect, because part c is not discussing the use of an inner class at all.

+ 0 Incorrect answer: "It only applies to events contained within the FooPanel class. If we have another class with the same event, it will not work." OR "You are not allowed to use the ActionListener outside of the FooPanel class". Neither of those is true. We "could" use an instance of this class to be an ActionListener for some other class' event. So it is

possible. But even if it were true that we couldn't (and that's NOT true), it is very unlikely that we would want to. An ActionListener in a GUI needs to make changes to the state of elements of a particular GUI. When you press a button, or scroll a scroller, or click a mouse, something is supposed to "happen". And making that thing happen likely requires access to the internal state of the FooPanel. And that fact pretty much negates the whole point you are trying to make, i.e. that "reuse" is something desirable here.

+ 0 Incorrect Ansswer: "If somewhere else in the program you need the same ActionListener, your code will not be DRY." That "seems" like a plausible answer---in general, we try to avoid duplicate code to make things DRY. But in this case, that's not the problem. . An ActionListener in a GUI typically needs to make changes to the state of elements of that GUI. When you press a button, or scroll a scroller, or click a mouse, something is supposed to "happen". And making that thing happen likely requires access to the internal state of the FooPanel. It is not likely that you are going to be able to reuse that code anywhere else in your program.

+ 0 Incorrect Answer: "This will make the FooPanel both extends JPanel and implements ActionListener, which may cause writing method of same name." So, I'm not sure what problem you are trying to describe here--- "may cause writing method of same name" isn't very precise. Same name as what? In any case having a class that both. extends JPanel and implements ActionListener is a perfectly legal thing to do, and causes no naming conflicts.

+ 2 Partial credit answer: "You have to create an instance of the FooPanel class itself within the FooPanel class. It's more complicated and its' not good for data encapsulation. There can only be one implementation." That last sentence, i.e. "there can only be one implementation" is what saved you from a zero credit answer---because that is true, though you didn't really explain why that's a problem. The first part is just false---you don't need a separate instance. The keyword "this" is a reference back to the instance itself. So, there is only one instance, and the data encapsulation is just fine---perfect, in a sense, since we are dealing only with a single object.

+ 0 Incorrect Answer. "This limits what the FooPanel class is able to do because each class is made so that it specializes at one job." This is incorrect. Implementing ActionListener doesn't "limit" what FooPanel can do. On the contrary, the essence of implementing an interface is that it provides a marker of some additional capability that the class CAN do. It expresses the idea "this class can do this thing, because it has all the necessary methods to do it". The second part of your sentence: "because each class is made so that it specializes at one job", describes an important design goal of a good object oriented system. But it is not the correct answer to THIS question.

+ 0 Incorrect answer: "There might not be a visual indicator of what or where the ActionListener is for the user". Incorrect because a "user" would never need to have any visual indicator of an ActionListener--its entirely an internal code construct, not a user-facing thing. If you mean "programmer" rather than "user", it's still incorrect. It will be visible from the fact that FooPanel class will say "implements ActionListener" right at the top, and the fact that the addActionListener() method for any widget with an ActionListener will take the parameter "this". Both of those are visual indicators of what or where the ActionListener is for the programmer.

+ 0 Not a rubric item, just a comment: see http://stackoverflow.com/questions/3124126/java-addactionlistenerthis for more information on this topic.

+ 0 Incorrect: "The FooPanel class can ONLY do what the ActionListener interface can do, no more, no less". OR "FooPanel would not be able to make method calls with a normal object functionality. FooPanel will only be used for waiting until the user makes an action. This means you cannot [illegible... have? hide?] an ActionEvent in this class definition." That is not correct. FooPanel can be a JPanel that has any

kind of functionality that a JPanel would normally have, PLUS it can also be an ActionListener. All we have to do is add an actionPerformed method to the class.

+ 3 PARTIAL CREDIT: "If FooPanel implements ActionListener, there can only ever be 1 interactive button/widget in the Panel..." That's not correct. There can be several. The problem isn't that we can't have more than one--its that we have to stuff all of that code for handling, say, a Button, a TextArea and a slider into a single method that has to complicated logic (e.g. a series of if tests) to determine which kind of event happened. Still, 3/5 because you are at least on the right track with this line of thinking.

+ 0 Incorrect: "That will let FooPanel implement a lot [sic] useless methods and in this case FooPanel needs to implement all the methods in ActionListener which are not what we want for FooPanel.". So, ActionListener has only one method (actionPerformed) and it really isn't a big deal to implement it. If we only need one action for a single widget, it may be less work to add this one method than to create a separate class.

1.4 (d) Disadvantage of separate class? (5 / 5)

+ 5 Correct: the separate class does not have access to the private members of FooPanel. (Which it will likely need in order to make things happen in the GUI).

+ 2 Partial credit: bulky code involved with separate classes for separate ActionListeners...vague or fails to mention idea that event handlers need access to instance variables.

+ 0 Incorrect: fails to identify a disadvantage of separate classes

+ 0 Left Blank

+ 3 Partial Credit: Correctly indicates that it is more work, with this architecture, to make changes to FooPanel from inside the ActionListener, but doesn't clearly explain why.

1.5 (e) Briefly describe third approach to ActionListener (not self, not separate class) (5 / 5)

+ 5 Correct: Making an inner class that implements ActionListener is the third approach.

+ 5 ALTERNATIVE correct answer--if "anonymous inner class" used as answer to part (e). This lists various advantages of an anonymous inner class.

+ 3 Partial credit: correctly identifies inner class, but doesn't mention that the inner class will be the one implementing ActionListener

+ 2 Partial credit: answer--"have a method in FooPanel implement ActionListener..." incorrectly says method instead of inner class, but interfaces must be implemented by a class

+ 0 Incorrect: doesn't identify a third approach on how to implement ActionListener

+ 0 blank

1.6 (f) Advantages of third approach (5 / 5)

+ 5 Correct: One or more inner class objects can be used to implement one or more ActionListeners, and each of those will have full access to the outer class' instance variables.

+ 5 ALTERNATIVE correct answer--if "anonymous inner class" used as answer to part (e). This lists various advantages of an anonymous inner class.

+ 3 Partial credit: correctly includes access to instance variables, but also should discuss how inner classes allow you to have multiple ActionListeners to handle multiple events

+ 3 Partial credit: correctly states that inner classes allow you to have multiple ActionListeners to handle multiple events, but doesn't say that inner classes have access to instance variables

+ 1 Partial credit: vague or incomplete response that fails to identify advantages of inner classes

+ 0 Incorrect: identifies disadvantages of inner classes

+ 0 Incorrect: doesn't identify an advantage of inner classes

+ 0 blank

QUESTION 2

Foo, Bar, Fum (inheritance) 20 pts

2.1 What methods can be invoked on foo? (5 / 5)

+ 5 **Correct: doThing() lines 5-7 (ok to say line 5 too).**

+ 0 OK to include main, since foo.main(args) will, technically, compile (I tried it). Also ok to omit this from answer, since we typically would not do this.

- 1 Directions indicated to include the line number in your answer.

2.2 What public data members can follow bar. (5 / 5)

+ 5 **Correct: mysteryBoolean and mysteryNumber**

- 2 -2 for including doThing() in the answer along with mysteryBoolean and mysteryNumber, since it is not a data member. Also, doThing() returns void, so an invocation of it cannot be used as argument of System.out.println()

+ 0 no points for answer doThing() if it is the only thing included in the answer. doThing() is not a data member.

+ 2 2 points for only including mysteryNumber, and not including mysteryBoolean.

+ 2 2 points for only including mysteryBoolean, and not including mysteryNumber.

2.3 Methods can be invoked on fum? (5 / 5)

+ 5 **[NOTE TYPO CORRECTION given to all students: "object foo" should read "object fum".] Correct: doThing, line 5 (or 5-7) and doOtherThing, line 30, (or 30-32)**

+ 0 Ok to include main; technically, it can be invoked on object fum; but not necessary for full credit.

+ 2 Partial Credit for mentioning only doOtherThing from lines 30-32 and leaving out doThing from lines 5-7, which is inherited.

2.4 Data members for fum? (5 / 5)

+ 5 **Correct: mysteryNumber and mysteryString**

- 2 -2 for including doThing and/or doOtherThing in the answer along with mysteryBoolean and mysteryString, since those are methods (member functions), not data members. Also, they return void, so an invocation of them cannot be used as argument of System.out.println()

+ 2 Partial credit for answer "mysteryString", omitting mysteryNumber which should also be included, as it is inherited.

+ 0 No credit for answer consisting only of doThing and doOtherThing, since those are methods (member functions), not data members. Also, they return void, so an invocation of them cannot be used as argument of System.out.println()

QUESTION 3

3 Two main categories of exceptions in Java. Names of two kinds, and reason why. (10 / 10)

+ 4 **4 of the 10 points are earned for correctly IDENTIFYING THE TWO CATEGORIES.. One category is Unchecked exceptions (which inherit from java.lang.RuntimeException) and the other is Checked exceptions, which inherit from java.lang.Exception. It is acceptable if the official names of these categories aren't given but it is clear that they are being referred to via the base class such exceptions have in common (RuntimeException / RuntimeExceptions without a space, Exception capitalized). "Thrown Exceptions" is not acceptable as all exceptions are thrown.**

+ 3 **3 of the 10 points are earned for correctly/completely identifying PURPOSE OF RUNTIME/UNCHECKED EXCEPTIONS. These indicate a problem with code logic. try/catch blocks for these are not required because they would clutter the code logic.**

+ 3 **3 of the 10 points are earned for correctly/completely identifying PURPOSE OF CHECKED EXCEPTIONS. These indicate a problem that the programmer cannot prevent, should anticipate, and should either handle with a try/catch block, or report to the caller by declaring that the method may throw the exception.**

+ 0 No points for incorrect explanation of Checked

exceptions.

+ 0 No points for incorrect explanation of Unchecked exceptions.

+ 0 Your explanation of the difference is that it is a distinction between catching the problem at compile time vs. run time, but that is not correct. Both of these happen at run time. The name is a bit misleading, I acknowledge---that's all the more reason to be sure you learn the correct distinction.

+ 2 Partial credit for an identification of the two categories that is incorrect, or only partially correct. Here are some examples of incorrect pairs of names: (1) IOExceptions vs. Exceptions (2) Runtime Exceptions vs. User-Defined Exceptions (3) runtime exceptions vs. undefined exceptions (4) RunTime Exceptions vs. Compiler Exceptions, (5) the first type is RuntimeExceptions, and the second type is the kind that uses the try/catch block (6) RuntimeException vs. CriticalException (7) RuntimeExceptoin and self-declared exceptions. Examples of correctly distinguishing are (1) checked vs. unchecked exceptions (2) Runtime exceptions vs. Checked Exceptions. (3) "RuntimeException" vs. "Regular plain-old Exception". (This last one isn't the "best" way to describe it, but I would accept as long as the explanations are correct.)

+ 2 Partial Credit for identifying the dichotomy between Runtime Exceptions and the other kind, but not indicating how the latter kind have to be handled in code (caught, or declared to be thrown.)

+ 0 Click here to replace this description.

QUESTION 4

4 Collection<? extends E> ... what does this mean? (5 / 10)

+ 10 Correct: c is an object of a class that implements the Collection interface, and it is a collection of either objects of type E, or of objects from classes or interfaces that extend (inherit from) type E. That explanation is sufficient for full credit... but to explain further with an example: if we have an instance of ArrayList<Student> and Undergrad extends Student, then c could be any of the following types: ArrayList<Student>, ArrayList<Undergrad>, PriorityQueue<Student>, PriorityQueue<Undergrad>, etc. Subject to deductions below. Note that E doesn't have to be a class. It could also be an interface.

+ 5 Partial Credit: Answer that is correct but incomplete--that says nothing wrong, but doesn't get to the point about what the <? extends E> part means.

+ 5 Partial Credit: "c is a collection of elements of any type... The reason that c can contain any type is because it is a collection of type E, which is a template parameter, meaning E can be of any type.". Yes, but no. E in this case is already bound. It is a specific type, the type that THIS ArrayList<E> instance is a collection of. <? extends E> actually RESTRICTS the type of what c can be a collection of. It restricts it to being only elements of type E, or any class/interface that extends E.

+ 5 Partial Credit "c is an instance of a class that implements an interface that extends the interface Collection<E>. " OR "any object of a class that extends classes that implement the Collection interface". NO.. If that were our intent, we might write <T extends Collection<E>> boolean addAll(T c). Or we could simply write: boolean addAll(Collection<E> c). Your answer seems to indicate that the ? extends is about extending Collection, but that is NOT the case. The extending is NOT of Collection, but of E itself.

- 2 -2 deduction from full credit answer for imprecise language: "elements that are of class E or extends class E as its parameter". What you mean to say is "elements that are of class/interface E, or any class/interface that extends E as its parameter". Those extra words are important to be precise in your meaning.

+ 5 Imprecise answer: "Collection<? extends E> c means that it could be a subclass of the given data type". This is too imprecise for full credit. What

does "it" refer to? Does it refer to c or E? I can't tell. And the given data type? What does that refer to? Collection<? extends E>? E? Without knowing this, I can't tell if your answer is correct or not. Therefore, it cannot earn full credit.

+ 5 Partial credit: Any answer that indicates that Collection<? extends E> c means that "c is any object that implements Collection, where it can be a collection of any type of object at all", i.e. there is no restriction on what type of object can be in the Collection. NO. E is already "bound" to whatever this PARTICULAR ArrayList<E> is a collection of. So, not ANY object. c is a Collection of objects of class or interface E, or any class or interface that extends E, where E is something *specific*, the SPECIFIC class or interface that this ArrayList is an ArrayList of. For example, if E is Animal, and Dog extends Animal then <? extends E> means that c can be a Collection<Animal> or a Collection<Dog>

+ 5 Partial Credit: E is of whatever type the ArrayList is. c is an interface that extends whatever type the ArrayList is. If it is an ArrayList<Integer> then c will be a Collection that extends Integer. NO.

+ 5 Partial credit: c is a type of Collection that extends E. NO. c is a type of Collection that contains inside it instances of E, or any class/interface that extends E. The Colection does NOT extend E. Not in any way.

+ 5 Partial credit: T"his means that the parameter c must be compatible with the elements of the ArrayList. By this, I mean that 'c' must be an instance of a class tht is the supertype E, or a subtype of E." Almost there, but NO. What you should have said is "This means that the parameter c __is a Collection of elements that___ must be compatible with ..." You left out those crucial words "is a Collection of elements that..."

+ 5 Partial credit: "c can be any class that extends E". c could also be an interface, and it must implement Collection interface

- 3 Collection is an interface so it must be implemented, it is not a super class that can be extended

+ 5 "The means that any object that extends Collection can be put in as an argument and that Collections ? will wrap around the object E so for example if ArrayList<Dog> is put as an argument then ? will wrap around Dog and allow the function to manipulate Dog objects." Incorrect. The "extends" keyword here is NOT about extending Collection. If ArrayList<Dog> were the context, than E is Dog, and what we have for c is a Collection of Dog or a collection of some object that extends Dog.

+ 9 9/10, for mostly correct answer. "This is a polymorphic argument. Essentially a polymorphic argument is when a super class is a parameter and one can pass a subclass through this parameter." [So far so good.] "In our case, addAll can take anything that is a subclass of collections, such as Stack, ArrayList, etc. [Ooops... not subclass. We noted that Collection is an interface, so we should have said anything that *implements* Collection. Continuing...] "The <? extends E> ensures that the argument is the same object or a subclass of the object that the ArrayList holds. YES.

+ 0 No credit: "This means that c can be any class that extends Object. In Java, primitives are not objects." That is incorrect. The type expression constrains the type of what c can be far more narrowly than "any class that extends Object", which is the same as saying "anything except a primitive". The statement "In Java, primitives are not objects" is correct, but that's not relevant to the problem, so no partial credit for that.

+ 5 Partial Credit: "It means that an ArrayList can add any object E or any collection of object E, or a subclass of E. The E object is any object or collection of objects that user defines it to be." Not exactly. We can't pass an object of type E or a subclass of E to this addAll method. We can only pass a Collection of objects of type E, or a Collection of a objects of some type that is a subclass of E.

+ 5 Incorrect/imprecise answer: "That this method works for all the classes that implement the collection interface. The type of collection must extend E..."

[Ooops. Not the type of collection (e.g. ArrayList, HashSet, PriorityQueue, Stack) must extend E, but the thing that c is a collection OF must be E or extend E. You are getting 5/10 for the benefit of the doubt that you meant to refer to the "thing that c is a Collection of", but your answer isn't precise enough to get full credit. Continuing...] " ... must be an element type (object references) but cannot be of primitive type. e.g. c cannot be of type ArrayList<int> but can be of type ArrayList<Integer>." So that last part is true, but "vacuously" true, since it is true of any parameterized type in Java. That part has nothing to do with the question asked, so you get no partial credit for it.
+ 5 Incorrect answer: "First, c is a specified collection, and inside the brackets, it stands for iterating all elements in this collection because addAll will append all of the elements." So, while it is true that addAll will append all of the elements, and c is a Collection, it is NOT true that the part in brackets (i.e. <? extends E> means to iterate over the collection.

1

Page: 1 Name: [redacted]

CS56—Midterm Exam

E02, W16, Phill Conrad, UC Santa Barbara

Monday, 02/29/2016

Name: [redacted]

Umail Ac[redacted]@ umail.ucsb.edu

- Pl[redacted]HE TOP OF EVERY PAGE
- Please put your pages **in order, facing the same way.**
  - All the odd pages have dots (•); these should be upper right, and facing up.
  - All the even numbered pages have crosses (×) at upper right and should be facing down.
- Be sure you turn in every page of this exam.
  - Each of the pages is numbered (e.g. Page 1, Page 2, etc.)
  - The last page clearly says "End of Exam".
- This exam is **closed book, closed notes, closed mouth, cell phone off**
- You are permitted **one sheet of paper** (max size 8.5x11") on which to write notes
- This sheet will be collected with the exam, and might not be returned
- Please write your name on your notes sheet

On your handout, on the back first method

The blacked out part should read:

boolean	add(E e)
	Appends the specified element to the end of this list.

Page: 2 N[redacted]

×

1. In Java Swing applications, we sometimes need an object that implements the `ActionListener` interface. Suppose that this situation arises in the context of a class called `FooPanel`.

   a. (5 pts) Given that we need an object that implements ActionListener inside of FooPanel, what is likely the purpose of class FooPanel, and why do we need an object that implements ActionListener inside of it?

   It is likely to impose widgets in a JFrame and we need an ActionListener to decide what those widgets do upon user input.

   b. (5 pts) There are three relationships that the object that implements ActionListener can have with the class FooBar. One of those, is that the object that implements ActionListener can be an instance of FooBar itself. In this case, what Java keyword is used to refer to the object that implements ActionListener?

   The Listener object, as in this case the user is the ActionListener.

   c. (5 pts) What is the main disadvantage of making the FooPanel class itself be the object that implements ActionListener?

   You are the actionListener and as such handling multiple buttons with different behaviours can be a hassle.

   d. (5 pts) A second technique is to make a completely separate class, separate from FooPanel, that implements ActionListener. What is the main disadvantage of this approach?

   You do not have access to all the instance variables in FooPanel.

   e. (5 pts) There is a third approach where the object that implements ActionListener has a different relationship with FooPanel from the two already described. What is this third approach? Briefly describe it.

   (Note: in Java 8, a fourth approach is to use Lambda Functions, but those are NOT covered on this exam, and it isn't what I'm looking for here. Those will be on the final exam.)

   It can be an inner class within the FooPanel class in which the inner class implements ActionListener.

   f. (5 pts) What are the advantages of this third approach to making an ActionListener over the other two already described?

   You have access to all the instance variables of the parent class and, for example, if you need multiple buttons with different behaviors it is simple to ~~mak~~ implement inner classes for each with their own unique actions.

2. Consider the code for classes Foo, Bar and Fum on the handout. Answer the questions below about this code.

a. (5 pts) Inside the `main` routine, locate the comment that says `This is line 14`. Suppose we were to invoke a method on the object referred to by reference `foo`. Disregarding methods that are ~~inherited~~ from class `java.lang.Object`, and considering only methods defined in the code here, list the methods we could invoke on object `foo`, and for each one, indicate the line number on which it is defined.

inherited

void doThing(), line 5

b. (5 pts) On line 13, there is a `System.out.println()` statement, with argument `"Hello"`. Suppose we were to replace the argument with a reference `bar`, the dot operator, and then any of the public data members that may be accessed through the reference bar. Disregarding any that might be inherited from class `java.lang.Object`, what is the complete list of data members that could follow `bar.` on this line? List them all.

public boolean mysteryBoolean, line 20
public int mysteryNumber, line 3

c. (5 pts) Inside the `main` routine, locate the comment that says `This is line 14`. Suppose we were to invoke a method on the object referred to by reference `fum`. Disregarding methods that are ~~inherited~~ from class `java.lang.Object`, and considering only methods defined in the code here, list the methods we could invoke on object ~~`foo`~~ `fum`, and for each one, indicate the line number on which it is defined.

inherited

void doThing(), line 5
void doOtherThing(), line 30

d. (5 pts) On line 13, there is a `System.out.println()` statement, with argument `"Hello"`. Suppose we were to replace the argument with a reference `fum`, the dot operator, and then any of the public data members that may be accessed through the reference fum. Disregarding any that might be inherited from class `java.lang.Object`, what is the complete list of data members that could follow `fum.` on this line? List them all.

public int mysteryNumber, line 3
public String mysteryString, line 28

3. (10 pts) Briefly describe the two main categories of exceptions in Java.

Be sure that your answer includes not only the names of the two kinds of exceptions, but also the reason that there are two different categories, and how they have to be treated differently.

Describe as if you were asked during a job interview. You should include enough detail so that the interviewer knows that you are very familiar with exceptions in Java, but not so much that you are wasting the interviewer's time.

There are two categories of exceptions in Java: RuntimeException and Exceptions. RuntimeExceptions do not have to be thrown or caught and they do not require the programmer to extend a class. This is the default exception designed to catch logical errors in your code ie: faulty logic.

Exceptions have to be caught or thrown and require the class to have "extends Exception". This is to catch anticipated errors the programmer may have such as a program being given a path to a nonexistant file.

4. (10 pts) On the reverse side of the handout, you will find the javadoc for the class `ArrayList<E>`. The third and fourth rows in this table contain the description of this method:

```
boolean addAll(Collection<? extends E> c)
```

I will tell you two additional pieces of information:

- that `Collection<E>` is an interface
- that a number of classes implement this interface, including `ArrayList<E>`, `HashSet<E>`, `PriorityQueue<E>`, and `Stack<E>`.

With that information, answer the following question.

The type of parameter `c` is given as `Collection<? extends E>`

What does this mean? Explain briefly.

This means that parameter c must be of a type that implements the Collection<E> interface with E being the type of object going into the collection ie: Dog.

CMPSC 56 E02

[redacted]

TOTAL POINTS

35 / 70

QUESTION 1

Java Swing 30 pts

1.1 (a) Purpose of FooPanel? (5 / 5)

+ 5 Correct: a widget that implements a GUI, or contains an interactive widget (e.g. a button) that needs a callback routine when it is clicked (or activated in some way).

+ 5 Alternative Correct Answer: to receive events and perform the desired actions in response.

+ 3 Partial Credit: Answer that is correct, but focuses on what an "interface" is, missing the point that ActionListener has a specific role in Java Swing. The question was getting at the "purpose" of class FooPanel in a specific sense, not an abstract sense.

+ 3 Partial Credit: Answer focuses on what an "inner class" is, missing the point that ActionListener has a specific role in Java Swing. The question was getting at the "purpose" of class FooPanel in a specific sense, not an abstract sense.

1.2 (b) What Java keyword? (0 / 5)

+ 5 Correct: this

+ 0 Incorrect: ActionEvent or ActionEvent object

+ 0 Incorrect: inner class

+ 0 Incorrect: Listener object

+ 0 Incorrect: public class FooPanel implements ActionListener

+ 0 Incorrect: Listener or Listeners

+ 0 Incorrect: Instance Variable

+ 2 Partial Credit: *this (The prefix * is a C++ thing.)

+ 0 incorrect: dot operator

+ 0 Incorrect: ActionListener

+ 0 Left Blank

+ 0 Incorrect: protected

+ 0 Incorrect: implements

+ 0 Incorrect: ActionHandler

+ 0 Incorrect: Event handler

+ 0 Incorrect: static

+ 0 Incorrect: widget

+ 0 Incorrect: new

+ 0 Not a rubric item, just a comment: see http://stackoverflow.com/questions/3124126/java-addactionlistenerthis for more information on this topic.

1.3 (c) Disadvantage of making FooPanel be object that implements ActionListener? (0 / 5)

+ 5 Correct: there can be only one actionPerformed method in the class. (That's a problem if you have more than two things you want to do---because you'll have to have complex code to handle all the different cases inside your one single actionPerformed method.)

+ 0 Left Blank

+ 4 PARTIAL CREDIT correct answer: It may not enforce separation of concerns or good object-oriented practices. [Awarded 4/5 because this is a good generic answer for why to make something a separate object. In this case, there is a more specific reason as well that is context dependent that we discussed in lecture.]

+ 0 Incorrect: Unable to reuse the code written in the inner class. This is incorrect, because part c is not discussing the use of an inner class at all.

+ 0 Incorrect answer: "It only applies to events contained within the FooPanel class. If we have another class with the same event, it will not work." OR "You are not allowed to use the ActionListener outside of the FooPanel class". Neither of those is true. We "could" use an instance of this class to be an ActionListener for some other class' event. So it is

possible. But even if it were true that we couldn't (and that's NOT true), it is very unlikely that we would want to. An ActionListener in a GUI needs to make changes to the state of elements of a particular GUI. When you press a button, or scroll a scroller, or click a mouse, something is supposed to "happen". And making that thing happen likely requires access to the internal state of the FooPanel. And that fact pretty much negates the whole point you are trying to make, i.e. that "reuse" is something desirable here.

+ 0 Incorrect Ansswer: "If somewhere else in the program you need the same ActionListener, your code will not be DRY." That "seems" like a plausible answer---in general, we try to avoid duplicate code to make things DRY. But in this case, that's not the problem. . An ActionListener in a GUI typically needs to make changes to the state of elements of that GUI. When you press a button, or scroll a scroller, or click a mouse, something is supposed to "happen". And making that thing happen likely requires access to the internal state of the FooPanel. It is not likely that you are going to be able to reuse that code anywhere else in your program.

+ 0 Incorrect Answer: "This will make the FooPanel both extends JPanel and implements ActionListener, which may cause writing method of same name." So, I'm not sure what problem you are trying to describe here--- "may cause writing method of same name" isn't very precise. Same name as what? In any case having a class that both. extends JPanel and implements ActionListener is a perfectly legal thing to do, and causes no naming conflicts.

+ 2 Partial credit answer: "You have to create an instance of the FooPanel class itself within the FooPanel class. It's more complicated and its' not good for data encapsulation. There can only be one implementation." That last sentence, i.e. "there can only be one implementation" is what saved you from a zero credit answer---because that is true, though you didn't really explain why that's a problem. The first part is just false---you don't need a separate instance. The keyword "this" is a reference back to the instance itself. So, there is only one instance, and the data encapsulation is just fine---perfect, in a sense, since we are dealing only with a single object.

+ 0 Incorrect Answer. "This limits what the FooPanel class is able to do because each class is made so that it specializes at one job." This is incorrect. Implementing ActionListener doesn't "limit" what FooPanel can do. On the contrary, the essence of implementing an interface is that it provides a marker of some additional capability that the class CAN do. It expresses the idea "this class can do this thing, because it has all the necessary methods to do it". The second part of your sentence: "because each class is made so that it specializes at one job", describes an important design goal of a good object oriented system. But it is not the correct answer to THIS question.

+ 0 Incorrect answer: "There might not be a visual indicator of what or where the ActionListener is for the user". Incorrect because a "user" would never need to have any visual indicator of an ActionListener--its entirely an internal code construct, not a user-facing thing. If you mean "programmer" rather than "user", it's still incorrect. It will be visible from the fact that FooPanel class will say "implements ActionListener" right at the top, and the fact that the addActionListener() method for any widget with an ActionListener will take the parameter "this". Both of those are visual indicators of what or where the ActionListener is for the programmer.

+ 0 Not a rubric item, just a comment: see http://stackoverflow.com/questions/3124126/java-addactionlistenerthis for more information on this topic.

+ 0 Incorrect: "The FooPanel class can ONLY do what the ActionListener interface can do, no more, no less". OR "FooPanel would not be able to make method calls with a normal object functionality. FooPanel will only be used for waiting until the user makes an action. This means you cannot [illegible... have? hide?] an ActionEvent in this class definition." That is not correct. FooPanel can be a JPanel that

has any kind of functionality that a JPanel would normally have, PLUS it can also be an ActionListener. All we have to do is add an actionPerformed method to the class.

+ 3 PARTIAL CREDIT: "If FooPanel implements ActionListener, there can only ever be 1 interactive button/widget in the Panel..." That's not correct. There can be several. The problem isn't that we can't have more than one--its that we have to stuff all of that code for handling, say, a Button, a TextArea and a slider into a single method that has to complicated logic (e.g. a series of if tests) to determine which kind of event happened. Still, 3/5 because you are at least on the right track with this line of thinking.

+ 0 Incorrect: "That will let FooPanel implement a lot [sic] useless methods and in this case FooPanel needs to implement all the methods in ActionListener which are not what we want for FooPanel.". So, ActionListener has only one method (actionPerformed) and it really isn't a big deal to implement it. If we only need one action for a single widget, it may be less work to add this one method than to create a separate class.

1.4 (d) Disadvantage of separate class? (5 / 5)

+ 5 Correct: the separate class does not have access to the private members of FooPanel. (Which it will likely need in order to make things happen in the GUI).

+ 2 Partial credit: bulky code involved with separate classes for separate ActionListeners...vague or fails to mention idea that event handlers need access to instance variables.

+ 0 Incorrect: fails to identify a disadvantage of separate classes

+ 0 Left Blank

+ 3 Partial Credit: Correctly indicates that it is more work, with this architecture, to make changes to FooPanel from inside the ActionListener, but doesn't clearly explain why.

1.5 (e) Briefly describe third approach to ActionListener (not self, not separate class) (2 / 5)

+ 5 Correct: Making an inner class that implements ActionListener is the third approach.

+ 5 ALTERNATIVE correct answer--if "anonymous inner class" used as answer to part (e). This lists various advantages of an anonymous inner class.

+ 3 Partial credit: correctly identifies inner class, but doesn't mention that the inner class will be the one implementing ActionListener

+ 2 Partial credit: answer--"have a method in FooPanel implement ActionListener..." incorrectly says method instead of inner class, but interfaces must be implemented by a class

+ 0 Incorrect: doesn't identify a third approach on how to implement ActionListener

+ 0 blank

1.6 (f) Advantages of third approach (1 / 5)

+ 5 Correct: One or more inner class objects can be used to implement one or more ActionListeners, and each of those will have full access to the outer class' instance variables.

+ 5 ALTERNATIVE correct answer--if "anonymous inner class" used as answer to part (e). This lists various advantages of an anonymous inner class.

+ 3 Partial credit: correctly includes access to instance variables, but also should discuss how inner classes allow you to have multiple ActionListeners to handle multiple events

+ 3 Partial credit: correctly states that inner classes allow you to have multiple ActionListeners to handle multiple events, but doesn't say that inner classes have access to instance variables

+ 1 Partial credit: vague or incomplete response that fails to identify advantages of inner classes

+ 0 Incorrect: identifies disadvantages of inner classes

+ 0 Incorrect: doesn't identify an advantage of inner classes

+ 0 blank

QUESTION 2

Foo, Bar, Fum (inheritance) 20 pts

2.1 What methods can be invoked on foo? (5 / 5)

+ 5 **Correct: doThing() lines 5-7 (ok to say line 5 too).**

+ 0 OK to include main, since foo.main(args) will, technically, compile (I tried it). Also ok to omit this from answer, since we typically would not do this.

- 1 Directions indicated to include the line number in your answer.

2.2 What public data members can follow bar. (0 / 5)

+ 5 Correct: mysteryBoolean and mysteryNumber

- 2 -2 for including doThing() in the answer along with mysteryBoolean and mysteryNumber, since it is not a data member. Also, doThing() returns void, so an invocation of it cannot be used as argument of System.out.println()

+ 0 **no points for answer doThing() if it is the only thing included in the answer. doThing() is not a data member.**

+ 2 2 points for only including mysteryNumber, and not including mysteryBoolean.

+ 2 2 points for only including mysteryBoolean, and not including mysteryNumber.

2.3 Methods can be invoked on fum? (5 / 5)

+ 5 **[NOTE TYPO CORRECTION given to all students: "object foo" should read "object fum".] Correct: doThing, line 5 (or 5-7) and doOtherThing, line 30, (or 30-32)**

+ 0 Ok to include main; technically, it can be invoked on object fum; but not necessary for full credit.

+ 2 Partial Credit for mentioning only doOtherThing from lines 30-32 and leaving out doThing from lines 5-7, which is inherited.

2.4 Data members for fum? (5 / 5)

+ 5 **Correct: mysteryNumber and mysteryString**

- 2 -2 for including doThing and/or doOtherThing in the answer along with mysteryBoolean and mysteryString, since those are methods (member functions), not data members. Also, they return void, so an invocation of them cannot be used as argument of System.out.println()

+ 2 Partial credit for answer "mysteryString", omitting mysteryNumber which should also be included, as it is inherited.

+ 0 No credit for answer consisting only of doThing and doOtherThing, since those are methods (member functions), not data members. Also, they return void, so an invocation of them cannot be used as argument of System.out.println()

QUESTION 3

3 Two main categories of exceptions in Java. Names of two kinds, and reason why. (2 / 10)

+ 4 4 of the 10 points are earned for correctly IDENTIFYING THE TWO CATEGORIES.. One category is Unchecked exceptions (which inherit from java.lang.RuntimeException) and the other is Checked exceptions, which inherit from java.lang.Exception. It is acceptable if the official names of these categories aren't given but it is clear that they are being referred to via the base class such exceptions have in common (RuntimeException / RuntimeExceptions without a space, Exception capitalized). "Thrown Exceptions" is not acceptable as all exceptions are thrown.

+ 3 3 of the 10 points are earned for correctly/completely identifying PURPOSE OF RUNTIME/UNCHECKED EXCEPTIONS. These indicate a problem with code logic. try/catch blocks for these are not required because they would clutter the code logic.

+ 3 3 of the 10 points are earned for correctly/completely identifying PURPOSE OF CHECKED EXCEPTIONS. These indicate a problem that the programmer cannot prevent, should anticipate, and should either handle with a try/catch block, or report to the caller by declaring that the method may throw the exception.

+ 0 **No points for incorrect explanation of Checked**

exceptions.

+ 0 No points for incorrect explanation of Unchecked exceptions.

+ 0 Your explanation of the difference is that it is a distinction between catching the problem at compile time vs. run time, but that is not correct. Both of these happen at run time. The name is a bit misleading, I acknowledge---that's all the more reason to be sure you learn the correct distinction.

+ 2 Partial credit for an identification of the two categories that is incorrect, or only partially correct. Here are some examples of incorrect pairs of names: (1) IOExceptions vs. Exceptions (2) Runtime Exceptions vs. User-Defined Exceptions (3) runtime exceptions vs. undefined exceptions (4) RunTime Exceptions vs. Compiler Exceptions, (5) the first type is RuntimeExceptions, and the second type is the kind that uses the try/catch block (6) RuntimeException vs. CriticalException (7) RuntimeExceptoin and self-declared exceptions. Examples of correctly distinguishing are (1) checked vs. unchecked exceptions (2) Runtime exceptions vs. Checked Exceptions. (3) "RuntimeException" vs. "Regular plain-old Exception". (This last one isn't the "best" way to describe it, but I would accept as long as the explanations are correct.)

+ 2 Partial Credit for identifying the dichotomy between Runtime Exceptions and the other kind, but not indicating how the latter kind have to be handled in code (caught, or declared to be thrown.)

+ 0 Click here to replace this description.

QUESTION 4

4 Collection<? extends E> ... what does this mean? (5 / 10)

+ 10 Correct: c is an object of a class that implements the Collection interface, and it is a collection of either objects of type E, or of objects from classes or interfaces that extend (inherit from) type E. That explanation is sufficient for full credit... but to explain further with an example: if we have an instance of ArrayList<Student> and Undergrad extends Student, then c could be any of the following types: ArrayList<Student>, ArrayList<Undergrad>, PriorityQueue<Student>, PriorityQueue<Undergrad>, etc. Subject to deductions below. Note that E doesn't have to be a class. It could also be an interface.

+ 5 Partial Credit: Answer that is correct but incomplete--that says nothing wrong, but doesn't get to the point about what the <? extends E> part means.

+ 5 Partial Credit: "c is a collection of elements of any type... The reason that c can contain any type is because it is a collection of type E, which is a template parameter, meaning E can be of any type.". Yes, but no. E in this case is already bound. It is a specific type, the type that THIS ArrayList<E> instance is a collection of. <? extends E> actually RESTRICTS the type of what c can be a collection of. It restricts it to being only elements of type E, or any class/interface that extends E.

+ 5 Partial Credit "c is an instance of a class that implements an interface that extends the interface Collection<E>. " OR "any object of a class that extends classes that implement the Collection interface". NO.. If that were our intent, we might write <T extends Collection<E>> boolean addAll(T c). Or we could simply write: boolean addAll(Collection<E> c). Your answer seems to indicate that the ? extends is about extending Collection, but that is NOT the case. The extending is NOT of Collection, but of E itself.

- 2 -2 deduction from full credit answer for imprecise language: "elements that are of class E or extends class E as its parameter". What you mean to say is "elements that are of class/interface E, or any class/interface that extends E as its parameter". Those extra words are important to be precise in your meaning.

+ 5 Imprecise answer: "Collection<? extends E> c means that it could be a subclass of the given data type". This is too imprecise for full credit. What

does "it" refer to? Does it refer to c or E? I can't tell. And the given data type? What does that refer to? Collection<? extends E>? E? Without knowing this, I can't tell if your answer is correct or not. Therefore, it cannot earn full credit.

+ 5 Partial credit: Any answer that indicates that Collection<? extends E> c means that "c is any object that implements Collection, where it can be a collection of any type of object at all", i.e. there is no restriction on what type of object can be in the Collection. NO. E is already "bound" to whatever this PARTICULAR ArrayList<E> is a collection of. So, not ANY object. c is a Collection of objects of class or interface E, or any class or interface that extends E, where E is something *specific*, the SPECIFIC class or interface that this ArrayList is an ArrayList of. For example, if E is Animal, and Dog extends Animal then <? extends E> means that c can be a Collection<Animal> or a Collection<Dog>

+ 5 Partial Credit: E is of whatever type the ArrayList is. c is an interface that extends whatever type the ArrayList is. If it is an ArrayList<Integer> then c will be a Collection that extends Integer. NO.

+ 5 Partial credit: c is a type of Collection that extends E. NO. c is a type of Collection that contains inside it instances of E, or any class/interface that extends E. The Colection does NOT extend E. Not in any way.

+ 5 Partial credit: T"his means that the parameter c must be compatible with the elements of the ArrayList. By this, I mean that 'c' must be an instance of a class tht is the supertype E, or a subtype of E." Almost there, but NO. What you should have said is "This means that the parameter c __is a Collection of elements that___ must be compatible with ..." You left out those crucial words "is a Collection of elements that..."

+ 5 Partial credit: "c can be any class that extends E". c could also be an interface, and it must implement Collection interface

- 3 Collection is an interface so it must be implemented, it is not a super class that can be extended

+ 5 "The means that any object that extends Collection can be put in as an argument and that Collections ? will wrap around the object E so for example if ArrayList<Dog> is put as an argument then ? will wrap around Dog and allow the function to manipulate Dog objects." Incorrect. The "extends" keyword here is NOT about extending Collection. If ArrayList<Dog> were the context, than E is Dog, and what we have for c is a Collection of Dog or a collection of some object that extends Dog.

+ 9 9/10, for mostly correct answer. "This is a polymorphic argument. Essentially a polymorphic argument is when a super class is a parameter and one can pass a subclass through this parameter." [So far so good.] "In our case, addAll can take anything that is a subclass of collections, such as Stack, ArrayList, etc. [Ooops... not subclass. We noted that Collection is an interface, so we should have said anything that *implements* Collection. Continuing...] "The <? extends E> ensures that the argument is the same object or a subclass of the object that the ArrayList holds. YES.

+ 0 No credit: "This means that c can be any class that extends Object. In Java, primitives are not objects." That is incorrect. The type expression constrains the type of what c can be far more narrowly than "any class that extends Object", which is the same as saying "anything except a primitive". The statement "In Java, primitives are not objects" is correct, but that's not relevant to the problem, so no partial credit for that.

+ 5 Partial Credit: "It means that an ArrayList can add any object E or any collection of object E, or a subclass of E. The E object is any object or collection of objects that user defines it to be." Not exactly. We can't pass an object of type E or a subclass of E to this addAll method. We can only pass a Collection of objects of type E, or a Collection of a objects of some type that is a subclass of E.

+ 5 Incorrect/imprecise answer: "That this method works for all the classes that implement the collection interface. The type of collection must extend E..."

[Ooops. Not the type of collection (e.g. ArrayList, HashSet, PriorityQueue, Stack) must extend E, but the thing that c is a collection OF must be E or extend E. You are getting 5/10 for the benefit of the doubt that you meant to refer to the "thing that c is a Collection of", but your answer isn't precise enough to get full credit. Continuing...] " ... must be an element type (object references) but cannot be of primitive type. e.g. c cannot be of type ArrayList<int> but can be of type ArrayList<Integer>." So that last part is true, but "vacuously" true, since it is true of any parameterized type in Java. That part has nothing to do with the question asked, so you get no partial credit for it.
+ 5 Incorrect answer: "First, c is a specified collection, and inside the brackets, it stands for iterating all elements in this collection because addAll will append all of the elements." So, while it is true that addAll will append all of the elements, and c is a Collection, it is NOT true that the part in brackets (i.e. <? extends E> means to iterate over the collection.

1

CS56—Midterm Exam

E02, W16, Phill Conrad, UC Santa Barbara

Monday, 02/29/2016

Name: [redacted]

Umail A [redacted]@ umail.ucsb.edu

- Please write your name **above AND AT THE TOP OF EVERY PAGE**
- Please put your pages **in order, facing the same way.**
  - All the odd pages have dots (•); these should be upper right, and facing up.
  - All the even numbered pages have crosses (×) at upper right and should be facing down.
- Be sure you turn in every page of this exam.
  - Each of the pages is numbered (e.g. Page 1, Page 2, etc.)
  - The last page clearly says "End of Exam".
- This exam is **closed book, closed notes, closed mouth, cell phone off**
- You are permitted **one sheet of paper** (max size 8.5x11") on which to write notes
- This sheet will be collected with the exam, and might not be returned
- Please write your name on your notes sheet

On your handout, on the back

first method

The blacked out part should read:

boolean	add(E e)
	Appends the specified element to the end of this list.

1. In Java Swing applications, we sometimes need an object that implements the `ActionListener` interface. Suppose that this situation arises in the context of a class called `FooPanel`.
   a. (5 pts) Given that we need an object that implements ActionListener inside of FooPanel, what is likely the purpose of class FooPanel, and why do we need an object that implements ActionListener inside of it?

   The purpose of FooPanel is to do something when the user makes an action within the panel. Without the Action Listener we cannot make any changes to the panel after an action is made.

   b. (5 pts) There are three relationships that the object that implements ActionListener can have with the class FooBar. One of those, is that the object that implements ActionListener can be an instance of FooBar itself. In this case, what Java keyword is used to refer to the object that implements ActionListener?

   Listener Object

   c. (5 pts) What is the main disadvantage of making the FooPanel class itself be the object that implements ActionListener?

   FooPanel would not be able to make method calls with a normal object functionality. FooPanel will only be used for waiting until the user makes an action this means you cannot have an action event in this class definition

   d. (5 pts) A second technique is to make a completely separate class, separate from FooPanel, that implements ActionListener. What is the main disadvantage of this approach?

   The main disadvantage is, we will have to include references to each class. Which will include getter and setter methods, to link those classes together.

   e. (5 pts) There is a third approach where the object that implements ActionListener has a different relationship with FooPanel from the two already described. What is this third approach? Briefly describe it.

   (Note: in Java 8, a fourth approach is to use Lambda Functions, but those are NOT covered on this exam, and it isn't what I'm looking for here. Those will be on the final exam.)

   Third approach is to have a method within FooPanel that implements action Listner and has an inline action event declaration

   f. (5 pts) What are the advantages of this third approach to making an ActionListener over the other two already described?

   this is easier to read, and the objects is more functional than the other two approaches.

2. Consider the code for classes Foo, Bar and Fum on the handout. Answer the questions below about this code.

methods foo

a. (5 pts) Inside the `main` routine, locate the comment that says `This is line 14`. Suppose we were to invoke a method on the object referred to by reference `foo`. Disregarding methods that are inherited from class `java.lang.Object`, and considering only methods defined in the code here, list the methods we could invoke on object `foo`, and for each one, indicate the line number on which it is defined.

inherited

foo: void doThingy(); //on line 5.

methods bar

b. (5 pts) On line 13, there is a `System.out.println()` statement, with argument `"Hello"`. Suppose we were to replace the argument with a reference `bar`, the dot operator, and then any of the public data members that may be accessed through the reference bar. Disregarding any that might be inherited from class `java.lang.Object`, what is the complete list of data members that could follow `bar.` on this line? List them all.

Bar: void doThingy(); //on line 22

methods fum

c. (5 pts) Inside the `main` routine, locate the comment that says `This is line 14`. Suppose we were to invoke a method on the object referred to by reference `fum`. Disregarding methods that are inherited from class `java.lang.Object`, and considering only methods defined in the code here, list the methods we could invoke on object ~~foo~~ fum, and for each one, indicate the line number on which it is defined.

inherited

fum: void doThingy(); //on line 5

void doOtherThingy(); //on line 30

Variables fum

d. (5 pts) On line 13, there is a `System.out.println()` statement, with argument `"Hello"`. Suppose we were to replace the argument with a reference `fum`, the dot operator, and then any of the public data members that may be accessed through the reference fum. Disregarding any that might be inherited from class `java.lang.Object`, what is the complete list of data members that could follow `fum.` on this line? List them all.

fum: public int mysteryNumber //on line 3

public String mysteryString // on line 28

3. (10 pts) Briefly describe the two main categories of exceptions in Java.

Be sure that your answer includes not only the names of the two kinds of exceptions, but also the reason that there are two different categories, and how they have to be treated differently.

Describe as if you were asked during a job interview. You should include enough detail so that the interviewer knows that you are very familiar with exceptions in Java, but not so much that you are wasting the interviewer's time.

Runtime exceptions: Exceptions that do not have to be caught or declared to be thrown.

User Defined exceptions: These are declared within a class, where we can extend any exception class and add new functionality to these already existing exception classes. these are helpful when debugging or showing an illegal Argument was entered, or a file was not read properly. We can use System.err() for these, so that we will also that the print statement of what happened.

4. (10 pts) On the reverse side of the handout, you will find the javadoc for the class `ArrayList<E>`. The third and fourth rows in this table contain the description of this method:

```
boolean addAll(Collection<? extends E> c)
```

I will tell you two additional pieces of information:

- that `Collection<E>` is an interface
- that a number of classes implement this interface, including `ArrayList<E>`, `HashSet<E>`, `PriorityQueue<E>`, and `Stack<E>`.

With that information, answer the following question.

The type of parameter `c` is given as `Collection<? extends E>`

What does this mean? Explain briefly.

whatever object "c" that is passed into addAll(c), will have already implemented every necessary method in Collection. Whenever implementing an interface we must ensure all methods to be implemented in fact are. So this means c will be treated just as any other collection but with new functionality.

CMPSC 56 E02a

[redacted]

TOTAL POINTS

30 / 30

QUESTION 1

Question 5 (coding) 30 pts

1.1 Correct structure (10 / 10)

+ 10 10 pts correct structure of class: public class TempSequence extends ArrayList<Integer> { ... } with two methods inside with correct method signatures. (Subject to deductions below)

- 3 Should not have a private instance variable of type ArrayList<Integer>, since we are using inheritance (as we did in lab03.) Therefore "this" is implicitly already an ArrayList<Integer>. You are using composition, not inheritance.
- 1 -1 missing close brace on class
- 5 Serious errors in class syntax.
- 2 There should not be a public data member of type double to store the average temperature. Instead, the variable to calculate the average shoud be a local variable of the averageTemp method. You are exposing a value that might not have a correct value, depending on whether the averageTemp method has been called recently or not. You should expose only the method UNLESS you have a way to ensure that the value being exposed is always correct---and since making it public makes it possible for someone outside the class to set it to any legal value of a double, you cannot ensure that.
- 2 TempSequence<> is incorrect. So is TempSequence<integer>. TempSequence is not a templated class.
- 3 It isn't necessary to implement a constructor, but if you do, you shouldn't do it by making a recursive call to the constructor inside the constructor you are defining. That will result in stack overflow, since there is no base case.
- 2 Keeping the avg as an instance variable is not appropriate, since you don't use it anywhere except inside the averageTemp method as a temporary result before returning the value. It should be a local variable inside that method.
- 3 Your overridden methods for add and size will lead to infinite recursion when called on temp inside averageTemp and aboveAverage.
- 3 Relying on an instance variable size is dangerous. The size could be changed by calls to the add or set methods. Instead, you should recompute size inside the averageTemp and aboveAverage methods.
- 3 You don't need a constructor, but if you implement one, it makes no sense to declare a local variable n of type ArrayList<Integer> inside it, that can't be accessed anywhere outside that constructor, and is never used.
- 1 Having an instance variable and a method with exactly the same name CAN be done, but SHOULD not be.
- 1 Since we are using inheritance not composition, you don't need a constructor. But since you wrote one, you need to at least write it correctly. You write list=new ArrayList<Integer>; It should be list=new ArrayList<Integer>();

1.2 public double averageTemp() method correct (10 / 10)

+ 10 Correct implementation (subject to deductions below.)

- 2 Since this is inheritance not composition, you should use this, not a private instance variable list inside the method.
- 3 Conversion to double must happen for either numerator or denominator BEFORE the division takes place. Otherwise, you get integer division and lose precision.

- 3 By initializing averTemp to get(0), and then STILL adding get(0) into the array, that value is being added into the array TWICE.
- 3 The double increment of i (both in loop header, and inside loop) will result in skipping every other element.
- 5 Did not accumulate a sum. Perhaps you meant to write avg+= instead of avg= ? In any case I wonder whether it's a good idea to accumulate a sum in a variable called avg.
- 3 Cannot use this[i] notation in Java for an ArrayList<Integer>. Must use this.get(i) instead
+ 0 This implementation is too far from correct to receive any partial credit. See submission specific comments below for why.
- 3 Divide by this.size() to get average, not by 2.
- 5 You are calculating your result based on the variable myTempSequence, which is a local variable declared inside the constructor; a variable which is out of scope and garbage collected immediately after the constructor ends. So this won't work at all. You should be using "this" instead of myTempSequence
- 2 this.length is a method for ArrayLists, and its called this.size() (javadoc was provided on the handout with the exam, so you didn't have to have this memorized, but you did need to look it up.)
- 1 Single letter variables names a, b, c are not good practice. How about len, sum and avg if you want something really short that is at least minimally descriptive?

1.3 public TempSequence aboveAverage() (10 / 10)

+ 10 Correct implementation (subject to deductions below.)

- 3 Since this is inheritance, not composition, you should use this, not a private instance of ArrayList<Integer> inside the method.
- 3 Value returned MUST be a TempSequence, NOT an ArrayList<Integer>
- 1 In Java, constructor must be invoked with (), i.e. TempSequence aboveList = new TempSequence() ;
- 1 Missing semicolon
- 1 Missing close brace on method.
- 3 TempSequence<Integer> is incorrect syntax; TempSequence is not a templated class.
- 3 You are using the average of an empty TempSequence as the basis of this method. How is that going to work?
- 3 Comparing against an instance variable that stores the average temperature requires that the averageTemp method has been called, and IT MIGHT NOT HAVE BEEN. So this code cannot be guaranteed to operate correctly without depending on a particular order of execution. Even setting it to a flag value such as -99999.99 initially, and recalling averageTemp if it doesn't match that value isn't guaranteed to work. Since TempSequence extends ArrayList<Integer>, there is the opportunity to add, remove, or set additional temperature values after averageTemp has always been called. The only way to ensure correctness is to ALWAYS call averageTemp inside aboveAverage, which then renders the instance variable useless.
- 3 CANNOT return 0 from a method that returns type TempSequence. You could just return the newly constructed empty TempSequence instead.
- 3 Rounding, as opposed to truncating, the average temperature to an integer, before comparing is not necessary, and introduces correctness problems. Suppose the average temperature ends up being 71.6 Then 72 is an above average temperature. However, 71.2 gets rounded up to 72 before the comparison and 72 will then not be included in the final result.
- 3 Invoking this.averageTemp() inside the loop results in recalculating that result every time. It would be better to calculate it only once outside the loop. Otherwise, the running time ends up being O(n^2).
- 3 Cannot use this[i] notation in Java for an ArrayList<Integer>. Must use this.get(i) instead and instead of result[j]=x use result.set(j,x) or result.add(x) instead.
- 3 In Java, declaring TempSequence result; creates an uninitialized reference to a TempSequence, not an

instance of TempSequence (as it would in C++). You need TempSequence result = new TempSequence();

+ 0 Regrettably, this is too far from a correct implementation to receive any partial credit. See the submission specific instructions for further notes.

- 2 newSeq.add(this.get(i)) NOT TempSequence.add(this.get(i)); add is a non-static method, so it has to be invoked on the instance (the instance of TempSequence that you are returning), not the class.

+ 0 Click here to replace this description.

- 3 NOT: results.get(count) = this.get(i); count++; BUT RATHER either: results.set(count,this.get(i)); count++ OR BETTER YET, JUST: results.add(this.get(i));

- 3 TempSequence<> is incorrect syntax, since TempSequence is not a templated class.

- 5 You are calculating your result based on the variable myTempSequence, which is a local variable declared inside the constructor; a variable which is out of scope and garbage collected immediately after the constructor ends. So this won't work at all.

Page: 1 Name [redacted]

CS56—Midterm Exam

E02, W16, Phill Conrad, UC Santa Barbara

Monday, 02/29/2016

Name [redacted]

Umail [redacted]@ umail.ucsb.edu

- Please write your name **above AND AT THE TOP OF EVERY PAGE**
- Please put your pages **in order, facing the same way.**
  - All the odd pages have dots (•); these should be upper right, and facing up.
  - All the even numbered pages have crosses (×) at upper right and should be facing down.
- Be sure you turn in every page of this exam.
  - Each of the pages is numbered (e.g. Page 1, Page 2, etc.)
  - The last page clearly says "End of Exam".
- This exam is **closed book, closed notes**, **closed mouth, cell phone off**
- You are permitted **one sheet of paper** (max size 8.5x11") on which to write notes
- This sheet will be collected with the exam, and might not be returned
- Please write your name on your notes sheet

On your handout, on the back first method

The blacked out part should read:

boolean	add(E e)
	Appends the specified element to the end of this list.

5

5. (30 pts) Write the full code for a public Java class named TempSequence that extends ArrayList<Integer> to represent a sequence of temperature readings. It should provide ~~one~~ two additional ~~method~~ methods beyond those inherited from ArrayList<Integer>
   - `public double averageTemp()` returns the average temperature across all readings in the ArrayList.
   - `public TempSequence aboveAverage()` returns a new TempSequence consisting of only the temperatures that were above the average of the temperatures in the TempSequence object on which the method was invoked.

```
public class TempSequence extends ArrayList<Integer>
{

    public double averageTemp(){
        double avg = 0.0;
        for(int i=0; i < this.size(); i++)
            avg += (double) this.get(i);
        avg = avg /((double) this.size());
        return avg;
    }
    public TempSequence aboveAverage(){
        TempSequence abav = new TempSequence();
        double avg = this.averageTemp();
        for(int i=0; i < this.size(); i++)
            if(avg < (double) this.get(i))
                abav.add(this.get(i));
        return abav;
    }
    public TempSequence(){
        super(); // ArrayList<Integer> constructor
    }
}
```

End of Exam

total points=100

CMPSC 56 E02a

[redacted]

TOTAL POINTS

26 / 30

QUESTION 1

Question 5 (coding) 30 pts

1.1 Correct structure (9 / 10)

+ 10 **10 pts correct structure of class: public class TempSequence extends ArrayList<Integer> { ... } with two methods inside with correct method signatures. (Subject to deductions below)**

- 3 Should not have a private instance variable of type ArrayList<Integer>, since we are using inheritance (as we did in lab03.) Therefore "this" is implicitly already an ArrayList<Integer>. You are using composition, not inheritance.

- 1 **-1 missing close brace on class**

- 5 Serious errors in class syntax.

- 2 There should not be a public data member of type double to store the average temperature. Instead, the variable to calculate the average shoud be a local variable of the averageTemp method. You are exposing a value that might not have a correct value, depending on whether the averageTemp method has been called recently or not. You should expose only the method UNLESS you have a way to ensure that the value being exposed is always correct---and since making it public makes it possible for someone outside the class to set it to any legal value of a double, you cannot ensure that.

- 2 TempSequence<> is incorrect. So is TempSequence<integer>. TempSequence is not a templated class.

- 3 It isn't necessary to implement a constructor, but if you do, you shouldn't do it by making a recursive call to the constructor inside the constructor you are defining. That will result in stack overflow, since there is no base case.

- 2 Keeping the avg as an instance variable is not appropriate, since you don't use it anywhere except inside the averageTemp method as a temporary result before returning the value. It should be a local variable inside that method.

- 3 Your overridden methods for add and size will lead to infinite recursion when called on temp inside averageTemp and aboveAverage.

- 3 Relying on an instance variable size is dangerous. The size could be changed by calls to the add or set methods. Instead, you should recompute size inside the averageTemp and aboveAverage methods.

- 3 You don't need a constructor, but if you implement one, it makes no sense to declare a local variable n of type ArrayList<Integer> inside it, that can't be accessed anywhere outside that constructor, and is never used.

- 1 Having an instance variable and a method with exactly the same name CAN be done, but SHOULD not be.

- 1 Since we are using inheritance not composition, you don't need a constructor. But since you wrote one, you need to at least write it correctly. You write list=new ArrayList<Integer>; It should be list=new ArrayList<Integer>();

1.2 public double averageTemp() method correct (7 / 10)

+ 10 **Correct implementation (subject to deductions below.)**

- 2 Since this is inheritance not composition, you should use this, not a private instance variable list inside the method.

- 3 **Conversion to double must happen for either numerator or denominator BEFORE the division takes place. Otherwise, you get integer division and lose precision.**

- 3 By initializing averTemp to get(0), and then STILL adding get(0) into the array, that value is being added into the array TWICE.
- 3 The double increment of i (both in loop header, and inside loop) will result in skipping every other element.
- 5 Did not accumulate a sum. Perhaps you meant to write avg+= instead of avg= ? In any case I wonder whether it's a good idea to accumulate a sum in a variable called avg.
- 3 Cannot use this[i] notation in Java for an ArrayList<Integer>. Must use this.get(i) instead
+ 0 This implementation is too far from correct to receive any partial credit. See submission specific comments below for why.
- 3 Divide by this.size() to get average, not by 2.
- 5 You are calculating your result based on the variable myTempSequence, which is a local variable declared inside the constructor; a variable which is out of scope and garbage collected immediately after the constructor ends. So this won't work at all. You should be using "this" instead of myTempSequence
- 2 this.length is a method for ArrayLists, and its called this.size() (javadoc was provided on the handout with the exam, so you didn't have to have this memorized, but you did need to look it up.)
- 1 Single letter variables names a, b, c are not good practice. How about len, sum and avg if you want something really short that is at least minimally descriptive?

1.3 public TempSequence aboveAverage() (10 / 10)

+ 10 Correct implementation (subject to deductions below.)

- 3 Since this is inheritance, not composition, you should use this, not a private instance of ArrayList<Integer> inside the method.
- 3 Value returned MUST be a TempSequence, NOT an ArrayList<Integer>
- 1 In Java, constructor must be invoked with (), i.e. TempSequence aboveList = new TempSequence() ;
- 1 Missing semicolon
- 1 Missing close brace on method.
- 3 TempSequence<Integer> is incorrect syntax; TempSequence is not a templated class.
- 3 You are using the average of an empty TempSequence as the basis of this method. How is that going to work?
- 3 Comparing against an instance variable that stores the average temperature requires that the averageTemp method has been called, and IT MIGHT NOT HAVE BEEN. So this code cannot be guaranteed to operate correctly without depending on a particular order of execution. Even setting it to a flag value such as -99999.99 initially, and recalling averageTemp if it doesn't match that value isn't guaranteed to work. Since TempSequence extends ArrayList<Integer>, there is the opportunity to add, remove, or set additional temperature values after averageTemp has always been called. The only way to ensure correctness is to ALWAYS call averageTemp inside aboveAverage, which then renders the instance variable useless.
- 3 CANNOT return 0 from a method that returns type TempSequence. You could just return the newly constructed empty TempSequence instead.
- 3 Rounding, as opposed to truncating, the average temperature to an integer, before comparing is not necessary, and introduces correctness problems. Suppose the average temperature ends up being 71.6 Then 72 is an above average temperature. However, 71.2 gets rounded up to 72 before the comparison and 72 will then not be included in the final result.
- 3 Invoking this.averageTemp() inside the loop results in recalculating that result every time. It would be better to calculate it only once outside the loop. Otherwise, the running time ends up being O(n^2).
- 3 Cannot use this[i] notation in Java for an ArrayList<Integer>. Must use this.get(i) instead and instead of result[j]=x use result.set(j,x) or result.add(x) instead.
- 3 In Java, declaring TempSequence result; creates an uninitialized reference to a TempSequence, not an

instance of TempSequence (as it would in C++). You need TempSequence result = new TempSequence();

+ 0 Regrettably, this is too far from a correct implementation to receive any partial credit. See the submission specific instructions for further notes.

- 2 newSeq.add(this.get(i)) NOT TempSequence.add(this.get(i)); add is a non-static method, so it has to be invoked on the instance (the instance of TempSequence that you are returning), not the class.

+ 0 Click here to replace this description.

- 3 NOT: results.get(count) = this.get(i); count++; BUT RATHER either: results.set(count,this.get(i)); count++ OR BETTER YET, JUST: results.add(this.get(i));

- 3 TempSequence<> is incorrect syntax, since TempSequence is not a templated class.

- 5 You are calculating your result based on the variable myTempSequence, which is a local variable declared inside the constructor; a variable which is out of scope and garbage collected immediately after the constructor ends. So this won't work at all.

1 [redacted]

CS56—Midterm Exam

E02, W16, Phill Conrad, UC Santa Barbara

Monday, 02/29/2016

Name: [redacted]

Umail Ad [redacted]mail.ucsb.edu

- Please write your name **above AND AT THE TOP OF EVERY PAGE**
- Please put your pages **in order, facing the same way.**
  - All the odd pages have dots (•); these should be upper right, and facing up.
  - All the even numbered pages have crosses (×) at upper right and should be facing down.
- Be sure you turn in every page of this exam.
  - Each of the pages is numbered (e.g. Page 1, Page 2, etc.)
  - The last page clearly says "End of Exam".
- This exam is **closed book, closed notes**, **closed mouth, cell phone off**
- You are permitted **one sheet of paper** (max size 8.5x11") on which to write notes
- This sheet will be collected with the exam, and might not be returned
- Please write your name on your notes sheet

On your handout, on the back first method

The blacked out part should read:

boolean	add(E e) Appends the specified element to the end of this list.

5

Page: 5 Name: [redacted]

5. (30 pts) Write the full code for a public Java class named TempSequence that extends ArrayList<Integer> to represent a sequence of temperature readings. It should provide ~~one~~ two additional ~~method~~ methods beyond those inherited from ArrayList<Integer>
   - `public double averageTemp()` returns the average temperature across all readings in the ArrayList.
   - `public TempSequence aboveAverage()` returns a new TempSequence consisting of only the temperatures that were above the average of the temperatures in the TempSequence object on which the method was invoked.

```
public class TempSequence extends ArrayList<Integer> {

    public TempSequence() {
        this();
    }
    public double averageTemp() {
        int sum = 0;
        for(int i=0; i < this.size(); i++) {
            sum += this.get(i);
        }
        double average = (double) (sum / this.size());
        return average;
    }
    public TempSequence aboveAverage() {
        double average = averageTemp();
        TempSequence above_average = new TempSequence();
        for (int i=0; i < this.size(); i++) {
            if (this.get(i) > average) {
                above_average.add(this.get(i));
            }
        }
        return above_average;
    }
}
```

End of Exam

total points=100

CMPSC 56 E02a

[redacted]

TOTAL POINTS

7 / 30

QUESTION 1

Question 5 (coding) 30 pts

1.1 Correct structure (7 / 10)

+ 10 **10 pts correct structure of class: public class TempSequence extends ArrayList<Integer> { ... } with two methods inside with correct method signatures. (Subject to deductions below)**

- 3 Should not have a private instance variable of type ArrayList<Integer>, since we are using inheritance (as we did in lab03.) Therefore "this" is implicitly already an ArrayList<Integer>. You are using composition, not inheritance.

- 1 **-1 missing close brace on class**

- 5 Serious errors in class syntax.

- 2 **There should not be a public data member of type double to store the average temperature. Instead, the variable to calculate the average shoud be a local variable of the averageTemp method. You are exposing a value that might not have a correct value, depending on whether the averageTemp method has been called recently or not. You should expose only the method UNLESS you have a way to ensure that the value being exposed is always correct---and since making it public makes it possible for someone outside the class to set it to any legal value of a double, you cannot ensure that.**

- 2 TempSequence<> is incorrect. So is TempSequence<integer>. TempSequence is not a templated class.

- 3 It isn't necessary to implement a constructor, but if you do, you shouldn't do it by making a recursive call to the constructor inside the constructor you are defining. That will result in stack overflow, since there is no base case.

- 2 Keeping the avg as an instance variable is not appropriate, since you don't use it anywhere except inside the averageTemp method as a temporary result before returning the value. It should be a local variable inside that method.

- 3 Your overridden methods for add and size will lead to infinite recursion when called on temp inside averageTemp and aboveAverage.

- 3 Relying on an instance variable size is dangerous. The size could be changed by calls to the add or set methods. Instead, you should recompute size inside the averageTemp and aboveAverage methods.

- 3 You don't need a constructor, but if you implement one, it makes no sense to declare a local variable n of type ArrayList<Integer> inside it, that can't be accessed anywhere outside that constructor, and is never used.

- 1 Having an instance variable and a method with exactly the same name CAN be done, but SHOULD not be.

- 1 Since we are using inheritance not composition, you don't need a constructor. But since you wrote one, you need to at least write it correctly. You write list=new ArrayList<Integer>; It should be list=new ArrayList<Integer>();

1.2 public double averageTemp() method correct (0 / 10)

+ 10 Correct implementation (subject to deductions below.)

- 2 Since this is inheritance not composition, you should use this, not a private instance variable list inside the method.

- 3 Conversion to double must happen for either numerator or denominator BEFORE the division takes place. Otherwise, you get integer division and lose

precision.
- 3 By initializing averTemp to get(0), and then STILL adding get(0) into the array, that value is being added into the array TWICE.
- 3 The double increment of i (both in loop header, and inside loop) will result in skipping every other element.
- 5 Did not accumulate a sum. Perhaps you meant to write avg+= instead of avg= ? In any case I wonder whether it's a good idea to accumulate a sum in a variable called avg.
- 3 Cannot use this[i] notation in Java for an ArrayList<Integer>. Must use this.get(i) instead
+ 0 This implementation is too far from correct to receive any partial credit. See submission specific comments below for why.
- 3 Divide by this.size() to get average, not by 2.
- 5 You are calculating your result based on the variable myTempSequence, which is a local variable declared inside the constructor; a variable which is out of scope and garbage collected immediately after the constructor ends. So this won't work at all. You should be using "this" instead of myTempSequence
- 2 this.length is a method for ArrayLists, and its called this.size() (javadoc was provided on the handout with the exam, so you didn't have to have this memorized, but you did need to look it up.)
- 1 Single letter variables names a, b, c are not good practice. How about len, sum and avg if you want something really short that is at least minimally descriptive?

This code does not reflect enough understanding of how Java works to receive any credit for this method.

Here's what this code does: it declares a local variable called avg.

Therre is then an assignment to a variable "list", but you didn't declare this variable, so that's a syntax error.

The line of code invokes the constructor of ArrayList<Integer>, so now list points to an empty list of integer values. It then tries to sum the values of that list--but note that there cannot possibly every be any values in that list. You just created it as a local variable, and did nothing to add anything into it. You then return that as the average, without dividing by the number of elements.

As much as it pains me to say it---if I were to give you full credit, and then make deductions for each of these errors, you'd get down to zero (or possibly less than zero, though I generally don't do that.) Which is why I'm simply not able to assign any partial credit.

1.3 public TempSequence aboveAverage() (0 / 10)

+ 10 Correct implementation (subject to deductions below.)
- 3 Since this is inheritance, not composition, you should use this, not a private instance of ArrayList<Integer> inside the method.
- 3 Value returned MUST be a TempSequence, NOT an ArrayList<Integer>
- 1 In Java, constructor must be invoked with (), i.e. TempSequence aboveList = new TempSequence() ;
- 1 Missing semicolon
- 1 Missing close brace on method.
- 3 TempSequence<Integer> is incorrect syntax; TempSequence is not a templated class.
- 3 You are using the average of an empty TempSequence as the basis of this method. How is that going to work?
- 3 Comparing against an instance variable that stores the average temperature requires that the averageTemp method has been called, and IT MIGHT NOT HAVE BEEN. So this code cannot be guaranteed to operate correctly without depending on a particular order of execution. Even setting it to a flag value such as -99999.99 initially, and recalling

averageTemp if it doesn't match that value isn't guaranteed to work. Since TempSequence extends ArrayList<Integer>, there is the opportunity to add, remove, or set additional temperature values after averageTemp has always been called. The only way to ensure correctness is to ALWAYS call averageTemp inside aboveAverage, which then renders the instance variable useless.

- 3 CANNOT return 0 from a method that returns type TempSequence. You could just return the newly constructed empty TempSequence instead.

- 3 Rounding, as opposed to truncating, the average temperature to an integer, before comparing is not necessary, and introduces correctness problems. Suppose the average temperature ends up being 71.6 Then 72 is an above average temperature. However, 71.2 gets rounded up to 72 before the comparison and 72 will then not be included in the final result.

- 3 Invoking this.averageTemp() inside the loop results in recalculating that result every time. It would be better to calculate it only once outside the loop. Otherwise, the running time ends up being O(n^2).

- 3 Cannot use this[i] notation in Java for an ArrayList<Integer>. Must use this.get(i) instead and instead of result[j]=x use result.set(j,x) or result.add(x) instead.

- 3 In Java, declaring TempSequence result; creates an uninitialized reference to a TempSequence, not an instance of TempSequence (as it would in C++). You need TempSequence result = new TempSequence();

+ 0 Regrettably, this is too far from a correct implementation to receive any partial credit. See the submission specific instructions for further notes.

- 2 newSeq.add(this.get(i)) NOT TempSequence.add(this.get(i)); add is a non-static method, so it has to be invoked on the instance (the instance of TempSequence that you are returning), not the class.

+ 0 Click here to replace this description.

- 3 NOT: results.get(count) = this.get(i); count++; BUT RATHER either: results.set(count,this.get(i)); count++ OR BETTER YET, JUST: results.add(this.get(i));

- 3 TempSequence<> is incorrect syntax, since TempSequence is not a templated class.

- 5 You are calculating your result based on the variable myTempSequence, which is a local variable declared inside the constructor; a variable which is out of scope and garbage collected immediately after the constructor ends. So this won't work at all.

- You are declaring a local variable of type ArrayList<Integer>, which is the wrong return type. A TempSequence "is-a" ArrayList<Integer>, but an ArrayList<integer> isn't a TempSequence, so it can't be returned from a method that is suppose to return one.

  Next, you create a local variable of type ArrayListInteger> called "list", but you didn't declare this variable, so that's a syntax error.

  As in the previous problem, list is now yet another completely empty ArrayList<Integer>. You then try to use [] notation on the ArrayList, which you cannot do in Java.

  You are comparing against the instance variable average---even supposing that averageTemp were implemented correctly, there is no guarantee that it has been called, because you never called it. So, there is no guarantee it doesn't still have the value 0.

  As much as it pains me to say it---if I were to give you full credit, and then make deductions for each of these errors, you'd get down to zero (or possibly less than zero, though I generally don't do that.) Which is why I'm simply not able to assign any partial credit.

1

Page: 1 Name: [redacted]

CS56—Midterm Exam

E02, W16, Phill Conrad, UC Santa Barbara

Monday, 02/29/2016

Name: [redacted]

Umail A[redacted]@ umail.ucsb.edu

- Please write your name **above AND AT THE TOP OF EVERY PAGE**
- Please put your pages **in order, facing the same way.**
  - All the odd pages have dots (•); these should be upper right, and facing up.
  - All the even numbered pages have crosses (×) at upper right and should be facing down.
- Be sure you turn in every page of this exam.
  - Each of the pages is numbered (e.g. Page 1, Page 2, etc.)
  - The last page clearly says "End of Exam".
- This exam is **closed book, closed notes, closed mouth, cell phone off**
- You are permitted **one sheet of paper** (max size 8.5x11") on which to write notes
- This sheet will be collected with the exam, and might not be returned
- Please write your name on your notes sheet

On your handout, on the back

first method

The blacked out part should read:

boolean	add(E e)
	Appends the specified element to the end of this list.

5

Page: 5 Name: [redacted]

5. (30 pts) Write the full code for a public Java class named TempSequence that extends ArrayList<Integer> to represent a sequence of temperature readings. It should provide ~~one~~ two additional ~~method~~ methods beyond those inherited from ArrayList<Integer>
   - `public double averageTemp()` returns the average temperature across all readings in the ArrayList.
   - `public TempSequence aboveAverage()` returns a new TempSequence consisting of only the temperatures that were above the average of the temperatures in the TempSequence object on which the method was invoked.

```
public class TempSequence extends ArrayList<Integer> {
  public double averageTemp() {
      double avg = 0.0;
      list = new ArrayList<Integer>();
      for(int i=0; i < list.length(); i++) {
          avg= avg + list[i];
      }
      average = avg;
      return avg;
  }

  public TempSequence aboveAverage() {
      ArrayList<Integer> tempnew = new ArrayList<>();
      list = new ArrayList<Integer>();
      for(int i=0; i < list.length(); i++) {
          if( list[i] > average) {
              tempnew.add(list[i]);
          }
      }
      return tempnew;
  }

  public double average = 0.0;
```

End of Exam

total points=100